# УПРАЖНЕНИЯ ПРОГРАММИРОВАНИЯ НА FORTH

Ф.Дж Гил Чика декабрь 2009 г.

## Глава 1

## Высказывания

### 1.1. Предыдущие соображения

Это сборник упражнений по программированию на Форте. Они варьируются от базовых манипуляций со стеком до числовых вычислений, и я думаю, что они иллюстрируют язык и то, как вы программируете на нем.

Что касается первого, то мы опустили некоторые более сложные темы, из которых можно было бы составить, например, упражнения, которые могли бы включать в себя использование immediate, postpone, catch и throw механик.

Что касается второго, то читатель сначала заметит,
что все упражнения мы составили без вмешательства переменных: все расчеты основаны на стеке. В целом это более элегантно и соответствует философии языка, хотя мы признаем, что некоторые упражнения были бы более простыми и понятными, если бы были введены локальные переменные. Но не забывайте, что это такое: упражнения.

Читатель также заметит, что слова в целом очень маленькие.
Если мы соберем словарь, содержащийся в этом тексте, и скомпилируем его (мы используем Gforth), то увидим, что среднее значение составляет менее 100 байт на слово.

Однако есть некоторые функции с очень линейной структурой, необычно длинные для языка, хотя и очень короткие по сравнению со средним размером функций в любом другом языке.

В этих немногих случаях очевидно, как их можно учитывать.

### 1.2. Обработка стека

Если не указано иное, предполагаются операции с целыми числами.

1. Напишите слово, которое производит эффект в стеке (ab -- a+1 b-1 a+b).
2. Напишите в стеке слово, которое производит эффект (ab -- a*a a*b).
3. Учитывая три числав стеке (a, b, c), вычислите d = b*b - 4ac.
4. Напишите слово, которое берет из стека два числа (a, b) и оставляет как результат (a - b) и (a + b).
5. Напишите слово, которое берет из стопки три числа (a, b, c)
и оставляет в результате a*a + b*b + c*c.
6. Напишите слово, которое берет из стека число n и оставляет его как
   n*n-5n + 6.
7. Напишите слово, которое берет адрес из стека и оставляет в ячейке тот же увеличенный адрес и значение, содержащееся в исходном адресе.
8. Напишите слово, которое берет два адреса из стека и оставляет оба адреса увеличенными на единицу, а также содержимое исходных адресов
( d1 d2 -- d1+1 d2+1 v1 v2).
9. Напишите слово, которое берет из стека два адреса и меняет их содержимое.
10. Напишите слово, которое производит эффект в стеке (a b>0 -- a*b) без использования умножения. Умножение - это не что иное, как многократное сложение.
11. Напишите слово, которое производит действие на стек (a b -- a/b a%b) , то есть принимает делимое и делитель и оставляет частное и остаток без использования деления. Деление — это не что иное, как многократное вычитание.
12. Реализуйте слова rot и -rot.
13. Реализуйте слово 2swap, эффект которого на стек будет (abcd -- cdab).
14. Реализуйте слово tuck, диаграмма стека которого имеет вид (ab -- bab).
15. Реализуйте 2drop, который удаляет два верхних элемента из стека. Реализуйте слово 2dup, чья стековая диаграмма имеет вид (ab -- abab). Запишите слово 2over , лияние которого на стек равно (abcd -- abcdab).

16. Реализуйте слово 3dup, эффект которого на стек равен (abc -- abcabc).
17. Напишите слово, которое удаляет каждый элемент из стека, только если в стеке есть элемент, который нужно удалить.
18. Напишите слово под названием Pick (его использование совершенно нежелательно), которое рассматривает стек как вектор, копируя n-й элемент, указанный наверху стека. n нумерует элементы стека от самого последнего, 0, до самого глубокого и не считает себя (например, если стек содержит 1, 2 и 3, результат выбора 1 2 3 2 будет 1 2 3 1)

## 1.3. Память

1. Напишите слово, которое восстанавливает значение, содержащееся в адресе памяти, оставляя в стеке исходный адрес, увеличенный на одну единицу (или в ячейке), и восстановленное значение.
2. Напишите слово, которое меняет местами значения, содержащиеся в двух адресах памяти.
3. Напишите слово, которое копирует в стек последний элемент из стека. возвратный стек.
4. Напишите слово, которое берет из стека адрес источника и адрес назначения, копирует значение, содержащееся в адресе источника, в адрес назначения и возвращает исходные адреса, увеличенные на одну единицу.
5. Напишите слово, которое берет из стека два адреса, один источник и один пункт назначения, и число n, копируя n байт из адреса источника в n коррелятивных позициях из адреса назначения.
6. Напишите слово, которое принимает из стека два адреса и возвращает: оба адреса увеличены на единицу и значения, содержащиеся в исходных адресах.
7. Напишите слово, которое переместит часть памяти на указанное количество позиций в сторону вверх. То же самое касается вниз.
8. Блок памяти задается базовым адресом и размером в байтах. Напишите два слова, которые ждут в стеке спецификации блока, и целое число, обозначающее количество байтов, которые блокируются.
блок необходимо переместить, и осуществить перемещение. Движение может быть выполнено вверх или вниз, отсюда и эти два слова.

## 1.4. Рациональные числа

1. Напишите слово, которое берет из стека два числа и оставляет у них наибольший общий делитель. Если a и b — эти числа и мы предполагаем, что a > b, используйте свойство mod(a, b) = mod(b, r), где r — остаток целочисленного деления a/b. Повторным применением этого свойства можно показать, что искомое значение соответствует значению первого аргумента, когда второй становится нулевым.

2. Два числа называются простыми, если наибольший общий делитель обоих равен 1. Напишите слово, которое принимает два числа и оставляет истинное или ложное значение в зависимости от того, простые они или нет.

3. Напишите слово, которое ищет минимальный делитель (отличный от 1) заданное число в стеке.

4. На основе предыдущего упражнения напишите слово, которое оставляет в стеке флажок, указывающий, является ли помещенное в него число простым или нет: (n - флаг).

5. Выведите все простые числа между двумя кубиками.

6. Напишите слово, производящее разложение на простые множители заданного числа.

7. Учитывая число n и основание b, найдите степень p такую,
 что b^p < n <= b^(p+1)

8. Учитывая целочисленную функцию с целочисленным аргументом f(n), найдите сумму n функций, начальное значение n от 0.
N и адрес f в стеке.

$$\sum_{i=0}^{N} f(i) \qquad (1.1)$$

9. Учитывая два целых числа в стеке, представляющие рациональное число (числитель, знаменатель), найдите наибольший общий делитель обоих и разделите их на него. Результатом является каноническая дробь исходного числа (например, 9/27 уменьшается до 1/3).

10. Напишите слова qrot, -qrot, qtuck и qnip, подходящие для манипуляций с рациональными числами (каждое рациональное число занимает две ячейки стека).

11. Рациональное число представляется двумя целыми числами, помещенными в стек: числителем и знаменателем.
Напишите слова для сложения, вычитания, умножения и деления рациональных чисел.

12. По двум числам в стеке n и k вычислите биномиальный коэффициент.

$$C_{n,k} = \frac{n!}{k!(n-k)!} \qquad (1.2)$$

13. Простой алгоритм генерации случайных чисел начинается с затравить n0 и вычислить согласно

$$n_{j+1} = (an_j + c)\,\%m \qquad (1.3)$$

В частности, когда a = 899, c = 0 и m = 32768, числа генерируются в интервале [0, 32767]. Напишите генератор случайных чисел.

14. Учитывая число p, найдите наибольшее
$n <= \sqrt{p}.$

15. По числу p и приближенному корню n <= p найти рациональное представление наилучшего приближения по формуле:

$$n_{k+1} = \frac{1}{2}\left(n_k + \frac{p}{n_k}\right) \qquad (1.4)$$

16. Учитывая целое число, напишите слово, которое печатает его представление в двоичном формате.

17. Учитывая целое число, напишите слово, которое печатает его представление в шестнадцатеричное.

18. Учитывая целое число n, напишите слово, которое запрашивает значение бита p, оставляя в стеке 0 или 1 в зависимости от того, равен ли указанный бит 0 или 1.

19. Учитывая целое число n и число p, напишите слова для активации и отключить бит p в n.

20. Комплексное число, состоящее из вещественной и мнимой целых частей, представлено в стеке двумя целыми числами: вещественной частью и мнимой частью. Напишите слова для сложения, вычитания, умножения и получения степени комплекса.

21. Учитывая два комплекса a + bj и c + dj, их соотношение определяется как

$$\frac{a+bj}{c+dj} = \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2}j \qquad (1.5)$$

Реализуйте слово, которое выполняет деление. Поскольку действительная и мнимая части являются уже не целыми числами, а рациональными, слово оставит в стеке сначала числители, действительную и мнимую, а затем знаменатель, общий.

## 1.5. Векторы

Для простоты мы рассматриваем векторы целых чисел. Версия для вещественных чисел очевидна. Мы ібудем полагаться на векторный тип. В письменной форме

```
10 vector X
```

Ожидается, что в словаре будет создан вектор из 10 элементов. Вектор из 10 (n) элементов занимает 11 (n+1) ячеек словаря, так как первая из них содержит размер вектора. Ожидается , что во время выполнения ранее созданное слово X оставит адрес первого элемента и количество элементов в стеке. Для диаграмм стека мы будем использовать буквы i для индекса, v для значения, d для адреса и N для количества элементов в векторе.

1. Напишите слово, вектор, которое создает вектор в словаре с количеством элементов, помещенных в стек, и объясненной структурой.

2. Напишите слово v! который ожидает в стеке значения, позиции,
в которой оно должно быть сохранено, размера вектора и его направления.
(например, 4 10 X v ! сохраняет значение 4 в позиции 10 вектора

3. Напишите слово v@ , которое ожидает в стеке элемент вектора, который вы хотите восстановить, его размер и адрес, и оставляет искомое значение в стеке.

4. Напишите слово, которое печатает элементы вектора в столбце. Ожидаются его размер и направление. Напишите слово, которое печатает диапазон элементов вектора.

5. Дан стек, содержащий ряд аргументов n1...np, их количесто m,
размер и направление вектора, написать слово, записывающее значения ni
последовательно с базового адреса.
Очевидно, что элементы вектора будут упорядочены обратно тому,
как они были сложены

6. Дан вектор, вычислите сумму всех его элементов.

7. Дан вектор, напишите слово, которое находит свое максимальное значение.
То же самое для минимального значения.

8. По заданному значению, содержащемуся в векторе, найти позицию, в которой оно находится.
находит.

9. По заданному значению вьяснить, лежит ли оно в заданном векторе.

10. Вычислите среднее значение вектора.

11. Напишите слово, которое заполняет все элементы вектора нулями. Обобщите его до любого значения, отличного от нуля.

12. Напишите слово, которое применяет любую функцию к каждому из
элементы вектора.

13. Напишите слово, которое записывает в каждый элемент вектора результат применения какой-либо функции к его индексу (например, каждый элемент i заменяется на f(i)).

14. Напишите функцию, заполняющую вектор случайными значениями, предоставленными функцией, разработанной в предыдущем разделе.

15. Напишите функцию, которая возвращает true или false в зависимости от того, находится ли значение между элементами вектора.

16. Учитывая вектор и два индекса, поменяйте местами соответствующие значения.

17. Напишите пузырьковый алгоритм для сортировки элементов вектор от меньшего к большему.

18. Напишите слово, записывающее в обратном порядке элементы a.
вектор.

19. Учитывая вектор, замените каждый элемент в позиции i суммой всех предыдущих элементов до i включительно.

1.6. Цепочки

Строка символов — это не что иное, как вектор символов. Если пространство зарезервировано для строки символов, первая ячейка используется для хранения зарезервированного размера. В отличие от того, что происходит с векторами, также интересно знать, каков эффективный размер хранимых цепочек. Поэтому мы определим строку как структуру, состоящую из трех полей: ячейки максимального размера; ячейка с занятым размером и последовательностью символов. Во время выполнения вызов строки должен оставить максимальный размер, эффективное число и адрес первого символа в стеке.

1. Введите слово, чтобы ссвдать строку символов.

2. Введите слово, чтобы вывести на экран строку символов.

3. Напишите слово, которое считывает символы1 клавиатуры! и формирует из них строку, хранящуюся в указанной строковой переменной.

4. Учитывая две строки, скопируйте первую во вторую.

5. Учитывая две строки, скопируйте вторую после первой, если это возможно.

6. Напишите слово, которое сравнивает две строки.

7. Напишите слово, которое сравнивает один и тот же интервал двух строк.

8. Напишите слово, которое проверяет, содержится ли данная строка (которую мы называем подстрокой) в другой. Напишите вторую версию предыдущего слова, которая вместо возврата флага возвращает -1, если подстрока не содержится в строке, или позицию первого символа подстроки внутри строки, если первый содержится во втором. Предполагается, что подстрока короче строки.

9. Напишите слова в заданную строку, преобразуйте ее текст в верхний регистр или строчная буква.

10.Напишите слово, которое возвращает позицию, в которой впервые появляется данный символ, или -1, если этот символ не встречается в строке.

11.Напишите слово, исключающее из строки символ, найденный в заданной позиции.

12.Напишите слово, чтобы вставить заданный символ в строку.
указанная позиция.

13.Напишите слово, в котором отсутствуют пробелы в начале строки.

14.Напишите слово, устраняющее найденные пробелы поезд в конце цепочки.

15.Строка состоит из слов, разделенных пробелами. Определите начальные адреса каждого слова.

16.Напишите слово, которое оставляет в стеке флаг, указывающий, есть ли данная строка пуста (содержит только пробелы) или нет.

17.Напишите слово, которое устранит лишние пробелы между словами. То есть он оставляет одно и только одно пустое пространство между слово и следующее.

18.Напишите слово, которое ожидает в стеке целое число, представляющее позиция внутри строки и начальный адрес этой строки, и оставить в стеке то количество символов, кроме пробела, которое между данной позицией и следующим пробелом есть. Псдхсдит для найти длину каждого слова в строке.

19.Дана строка, напишите слова, чтобы i обернуть ее влево и вправо.
и центр. Понятно, что регулировка происходит без изменения длины! эффективная цепочка.

1.7 Матрицы

Вектора достаточно для представления матрицы двух или более измерений. Однако желательно скрыть преобразование между координатами (строка, столбец) и линейное направление внутри вектора, а также удобно, что Тип данных matrix имеет связанную информацию о количестве строк и столбцов, поскольку тот же вектор из 18 элементов также может представляют собой матрицу размером 3x6 и один из элементов 2x9. Для простоты будем рассматривать массивы целых чисел. Версии можно писать без труда для действительных или рациональных матриц.

1. Напишите слово, позволяющее создать матрицу. Размеры Он/1 будут предварительно помещены! в стек. В словаре место будет зарезервировано для этих двух чисел, а затем для суммы предметы. Во время выполнения адрес первого элемента будет помещен в стек. Для получения количества строк и количества столбцов матрицы будут записаны конкретные слова.

2. Напишите слово который ожидает в стеке сохранения значения, пары (i, j) и массива и сохраняет указанное значение в соответствующей позиции массива.

3. Напишите слово m@, которое принимает значение строки и столбца (i, j) и массив из стека и извлекает соответствующий элемент.

4. Напишите слово. который печатает элементы матрицы на экране, разделяя строки на разные строки.

5. Напишите слово, которое заполняет массив значением 0.

6. Напишите слово, относящееся к каждому элементу матрицы x.
функция f(x)

7. Напишите слово, которое ожидает в стеке массив и адрес функции f(i, j), и записывает значение f(i, j) в каждый элемент (i, j) массива.

8. Напишите слово, которое ожидает в стеке матрицу и возвращает максимальное значение, а также строку и столбец, в которых оно находится. То же самое и с минимальным значением.

9. Напишите слово, заполняющее массив случайными значениями.

10. Напишите слова, чтобы поменять местами две строки и два столбца одного.штаб.

11.Напишите слово, переставляющее матрицу (квадрат), поменяйте местами каждый элемент (i, j) для элемента (j, i).

12.Напишите слово, умножающее строку матрицы на заданное значение. Значение, которое нужно умножить, строка и матрица ожидаются в стеке. То же самое для столбца.

13.Напишите слово, заменяющее одну строку в результате добавления к ней другой. То есть, если добавляемые строки — это i и j, для каждого столбца k aik = aik + ajk.

1.8. Численные вычисления

В этом разделе предполагаются действительные числа.

1. Дана реальная функция реальной переменной, напишите слово, которое печатает на экране таблицу значений этой функции. Приращение аргумента, нижний и верхний пределы интервала и адрес функции ожидаются в стеке.

2. Напишите слова для создания векторов действительных чисел, восстановите элементы и сохранять значения в заданных позициях.

3. Напишите слово, которое ожидает в стеке нижнего значения независимой переменной, приращения этой переменной, адреса функции и вектора действительных чисел. Сохраните в этом векторе значение функции для последовательных значений независимой переменной, разделенных установленным приращением, то есть, если a — начальное значение независимой переменной, a h — приращение, вектор будет хранить ' af (a) , f(a + h), f(a + 2h) и т. д.

4. Учитывая набор вещественных значений x0  xn, целое число n+1 и вектор вещественных значений, напишите слово, записывающее элементы xi в последовательные позиции вектора.

5. Вычислите показательную функцию для аргумента x, используя ряд

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots \qquad (1.6)$$

6. Даны функция f(x) и два значения x = a и x = b такие, что существует корень f(x) = 0 в интервале [a, b], найти указанный корень методом деления пополам. .

7. Используйте предыдущее упражнение, чтобы написать слово, которое принимает интервал действительной прямой и направление функции и возвращает все действительные корни функции в заданном интервале.

8. Напишите версию, использующую вещественные числа алгоритма, описанного в разделе о целых числах, для вычисления квадратного корня из x >= 1.

9. Дан вектор, два аргумента a < b и адрес функции:
вычислить интеграл

$$\int_a^b f(x)dx \qquad (1.7)$$

методом трапеций с разрешением, указывающим размерность вектора.

10. Учитывая два действительных вектора, хранящие ряды значений xi и yi , вычислите коэффициенты (a, b) линии y(x) = ax+b, которая лучше всего соответствует данным (xi , yi) (подгонка m ' наименьшие квадраты).

11. Напишите слово, которое по заданному в стеке значению, принадлежащему области определения функции и самой функции, численно вычисляет значение производной функции в данной точке.

12. Мы можем представить многочлен вектором, хранящим свои коэффициенты при возрастающих степенях независимой переменной. Например, вектор [ 0.2e 0.8e -1. 3e ] представляет полиномиал p(x) = 0,2 + 0,8x - 1,3x значение x и вектор, который представляет  2. Напишите слово, которое ожидает полином и вычисляет значение полинома в точке.

13. Рассматривая элементы вещественного вектора как коэффициенты полинома, перезапишите его производной.

14. То же, но с интегралом.

15. Дан вектор, содержащий вещественные коэффициенты многочлена p(x) и два числа (a, b), получим

$$\int_a^b p(x)dx \qquad (1.8)$$

16. Проинтегрируем простое дифференциальное уравнение методом Рунге-Кутты четвертого порядка:

$$\frac{dx}{dt} = f(t) \qquad (1.9)$$

заданные начальные значения (t0, x0).

17. Проинтегрируем простое дифференциальное уравнение методом Рунге-Кутты четвертого порядка:

$$\frac{dx}{dt} = f(x) \qquad (1.10)$$

18. Двойной интеграл

$$\int_0^1 \int_0^1 f(x,y)dxdy \qquad (1.11)$$

может быть решена приближенно методом Монте-Карло.
Если в интервале [0, 1] генерируется большое количество N пар случайных чисел (xi , yi) , интеграл приблизительно равен

$$\frac{1}{N}\sum_i f(x_i, y_i) \qquad (1.12)$$

Напишите слово, которое принимает направление f(x, y) и возвращает значение интеграла.

1.9. Файлы

Имея дело с записью и чтением числовых значений в файлах, мы будем считать, что эти числа являются целыми числами. Мы также будем предполагать, что файлы являются текстовыми, а не двоичными, поэтому числовые значения сохраняются в виде строк, представляющих числа. Строковые слова, разработанные ранее, будут полезны.

1. Напишите слово, которое открывает текстовый файл, читает его построчно и печатает каждую строку на экране.

2. Напишите версию предыдущего слова с нумерацией каждой строки перед ее печатью.

3. Напишите слово, которое принимает строки с клавиатуры, заканчивающиеся на клавишу Enter и записывает их в файл.

4. Файл состоит из одного столбца целых чисел. Напишите слово, которое считывает их одно за другим и заполняет ими вектор.

5. То же, но с двумя столбцами, так что первый столбец
заполняет один вектор, а второй другой.

6. Напишите слово, которое читает все строки файла и печатает те, которые содержат заданную последовательность символов.

7. Дан текстовый файл, состоящий из ряда столбцов. Напишите слово, которое берет из стека числа в двух из этих столбцов и выводит их на экран.

8. Предположим, текстовый файл содержит столбцы. Сохраните максимальную ширину каждого столбца в векторе.

9. Используйте предыдущий результат, чтобы, имея файл, который содержит столбцы, ширина каждого столбца варьируется от одной строки к другой, взять файл и создать другой, в котором ширина каждого столбца будет одинаковой.

## 1.10. Клавиатура и экран

Очевидно, что этот раздел зависит от системы, под которой вы работаете.
Мы разработали его для терминала Linux.

1. Напишите слово, которое, учитывая символ и координаты строки и столбца в стеке, печатает этот символ на экране в указанной позиции.

2. Напишите слово, которое печатает на экране строку с числом.
заданное количество копий указанного символа.

3. Напишите слово, которое печатает таблицу с символами ASCII.

4. Напишите слово, которое рисует на экране прямоугольник заданной ширины и высоты.
Символы «-» будут использоваться для обводок горизонта, для вертикальных и «+» в стеке для углов. Координаты верхнего левого угла, ширины и высоты ожидаются «—».

5. Напишите программу, которая, учитывая символ «*» в центре поля, нарисованного предыдущим словом, использует клавиши «A», «S», «W», «Z» для перемещения его влево и вправо. , вверх и вниз, не выходя за пределы коробки. Используйте клавишу «Q», чтобы выйти из программы.

6. Напишите слово, которое отправляет на терминал последовательность ANSI, активирующую обратное видео.

7. Напишите слово, которое заполняет строку из 16 символов [A,...,O,-]. Первые 15 будут выбраны в случайном порядке. Цель состоит в том, чтобы нарисовать на экране матрицу, подобную той, что изображена на рисунке:

```
+---+---+---+---+
| G | B | I | J |
+---+---+---+---+
| M | H | L | C |
+---+---+---+---+
| N | A | E | D |
+---+---+---+---+
| F | O | K | - |
+---+---+---+---+
```

и играйте с клавиатурой, перемещая буквы, используя пробел , обозначенный знаком «-», пока не переставите их в алфавитном порядке.

8. Учитывая символ в стеке, напишите программу, которая выводит на экран макро версию этого символа размером 10 строк на 8 столбцов. Например, для символа «P»

```
##########
##      ##
##      ##
##      ##
##      ##
##########
##
##
##
##
```

9. Напишите процедуру, которая печатает строку в верхней части экрана. Далее первый символ строки будет отмечен символом «*», который будет печататься в нижней строке. Клавиши «A» и «S» будут использоваться для перемещения вперед и назад по

строке, обновления сигнала «*» и печати указанного символа в формате macro , как показано на рисунке.
Многие программисты игнорируют существование Форта. *

```
##########
##      ##
##      ##
##      ##
##      ##
##########
##
##
##
##
```

1.11. Структуры данных

1.11.1. Стеки и очереди

1. Пользователь может создать стек для собственного использования, выделив набор ячеек, первая из которых является индексом, указывающим следующую свободную позицию в стеке, а вторая — максимальное количество элементов, которые он может содержать.
Напишите слово для создания пользовательских стеков в форме

20 стопок

Во время создания резервируется пространство, а значение первой ячейки устанавливается равным 1 (индекс 0). Во время выполнения адрес пользовательского стека помещается в стек. Стек — это не что иное, как вектор, используемый определенным образом.

2. Запишите слова p+ и p- для сложения и разложения значений в стеке пользователя.

3. Напишите слова p-drop , p-swap и p-dup, которые действуют на стек. пользователя.

4. Напишите слова pr> и pr< , которые действуют в пользовательском стеке. Напишите слова, которые переносят элементы между двумя пользовательскими стеками, извлекая элемент из одного стека и помещая его в другой.

5. Первая ячейка стека используется как индекс, который увеличивается и уменьшается по мере того, как элементы складываются или разбираются. В очереди FIFO первым извлекается не последний элемент в стопке, а первый. Предметы складываются сверху , но удаляются снизу . Теперь, чтобы быстро не исчерпать место в очереди, при извлечении элемента необходимо будет переместить вниз все остальные. Поскольку это дорого, предпочтительно реализовать его в виде кольцевой очереди. Циклическая очередь — это вектор, первые две ячейки которого зарезервированы. В первом хранится индекс следующей свободной позиции для постановки в очередь. Во-вторых, позиция следующего элемента, который будет исключен из очереди.

После каждой операции постановки в очередь или удаления из очереди соответствующий индекс увеличивается. Если очередь имеет N элементов, от 0 до N - 1, когда индекс действителен, а N -1 установлено в 2. Очередь заполнена, когда индекс записи находится сразу за индексом чтения, и пуста, когда индекс чтения равен 2. сразу за индексом записи. В нашей реализации мы зарезервируем не две, а четыре ячейки: указатели чтения и записи, количество элементов, которые содержит очередь в каждый момент времени, и максимальное количество элементов.

Напишите слово, чтобы создать пользовательскую очередь FIFO, резервируя заданное количество ячеек,

20 fifo Z \ создает очередь FIFO из 4+16 элементов.

6. Напишите набор слов, позволяющий управлять очередью FIFO.

### 1.11.2. Дружественная динамическая память

Многие структуры данных, такие как списки и деревья, основаны на указателях, которые, в свою очередь, используют динамическую память. Такие языки, как C и Pascal, позволяют объявлять указатели, выделять для них память и освобождать эту память, когда она больше не нужна. Другие языки, такие как Java или Lisp, широко используют указатели, но они не являются особенностью языка, а скорее одним из элементов, посредством которых язык реализуется. Вся сложность скрыта от программиста. Точно так же вся гибкость и мощь также остаются недоступными.

Все системы предлагают динамическую память, но здесь мы будем использовать другой подход: программа может зарезервировать часть словаря и динамически управлять ею, захватывая и освобождая память при необходимости. Но не напрямую, а через менеджер памяти, предлагающий интерфейс и состоящий из необходимых слов для внутреннего управления выделением и освобождением памяти, ее дефрагментацией, управлением ошибками и т.п.

В англоязычной литературе участок памяти, который будет управляться динамически, захватывая или освобождая его части, называется кучей, что иногда переводится как куча. Нам более подходящими кажутся такие названия, как склад или склад. Мы назовем это вторым способом.
Существует множество способов управления складом, и можно обратиться к любой книге по операционной системе. Наш подход будет очень и очень простым: и понять, и реализовать. Однако в итоге мы получим настоящий менеджер динамической памяти и ничего больше. Таким образом, этот раздел содержит одно упражнение, которое будет решено с помощью набора слов, которые мы будем объяснять по мере их написания. Таким образом, это не просто упражнение, это небольшой проект, в котором станет очевидной собственная восходящая методология Форта.

1. Напишите менеджер динамической памяти, который резервирует хранилище и управляет им. Программа должна иметь возможность объявить хранилище, указав его имя и размер, которым будет управляться:

10000 хранилища A.

## Глава 2 Решения

Что касается предлагаемых решений, будут приняты во внимание некоторые вещи:

а) решения не уникальны;
б) что из возможных наших решений упражнения было бы разумным, но мы сомневаемся, что оно оптимально;
в) некоторые слова, безусловно, можно было бы учесть.

Что касается обозначений, в комментариях стека используются буквы для разных типов данных:

a,b,c,m,n,k для целых чисел;
d,e для адресов памяти;
u,v для значений, полученных из адресов памяти;
x,y,z для действительных чисел;
xt для токена выполнения слова;
f — флаг или результат теста (истина или ложь);
p,q,r — рациональные или комплексные числа, каждое из которых представляет два целых числа в стеке:
(числитель, знаменатель) или (действительная часть, мнимая часть).

### 2.1. Обработка стека

```
1. : p1 1- swap 1+ swap 2dup + ;
2. : p2 over * swap dup * swap ;
3. : p3 swap dup * -rot 4 * * - ;
```

```
4. : p4 2dup - -rot + ;
5. : p5 dup * swap dup * + swap dup * + ;
6. : p6 dup dup * swap 5 * - 6 + ;
7. : p7 dup @ swap cell+ swap ;
8. : p8 2dup @ swap @ swap >r >r 1+ swap 1+ swap r> r> ;
9. : p9 2dup @ swap @ rot ! swap ! ;
10 : p10 0 swap 0 do over + loop nip ;

11. : p11 0 -rot
      begin
           2dup >=
      while
           tuck - swap rot 1+ -rot
      repeat
      drop ;

12. : rot ( a b c -- b c a) >r swap r> swap ;
    : -rot ( a b c -- c a b) swap >r swap r> ;

13. : 2swap ( a b c m -- c m a b) >r -rot r> -rot ;
14. : tuck ( a b -- b a b) dup -rot ;
15.  : 2drop ( a b -- ) drop drop ;
      : 2dup ( a b -- a b a b) over over ;
      : 2over ( a b m n -- a b m n a b )
        2swap 2dup >r >r 2swap r> r> ;

16.  : 3dup ( a b c -- a b c a b c)
        >r 2dup r> dup >r -rot r> ;
17. : cls depth dup 0= if drop else 0 do drop loop then ;
```

18. Очевидно, что элементы, находящиеся выше того, из которого вы хотите извлечь копию, придется переместить в другие места, а затем вернуть. Куда они могли переехать? Одним из мест может быть стек возврата. Но поскольку операция удаления n элементов должна состоять из цикла, удаляющего по одному элементу за раз, и этот цикл должен иметь стек возврата для выполнения, этот стек не кажется лучшим местом, хотя его можно использовать. Более естественно использовать словарь, начиная с первой свободной позиции, адрес которой возвращает слово HERE. Таким образом, выборка по существу состоит из двух шагов: а) удаление n элементов из стека и их последовательное размещение, начиная с первой свободной позиции в словаре, и б) восстановление этих элементов по одному или одному, перенося на каждом этапе копию элемента, который вы хотите выполнить репликацию на вершину стека, чтобы в конце второго цикла он оказался поверх всех остальных. Remove-n ожидает в стеке адрес, по которому можно поместить элементы, и число элементов, которые вы хотите переместить. Они вернут адрес, с которого нужно начать, и количество элементов, которые вы хотите вернуть в стек (в порядке убывания от начального адреса).

```
: cell+ 4 + ; \ una celda tiene 4 bytes
: cell- 4 - ;
: quitar-n ( d n -- d') 0 do 2dup ! nip cell+ loop ;
: devolver-n ( d n -- d') 0 do dup @ -rot cell- loop ;
: pick dup >r here swap quitar-n
cell- over swap r> devolver-n drop ;
```

2.2. Память

```
1.
: @+c ( d -- v d') dup @ swap cell+ swap ;
: @+1 ( d -- v d') dup c@ swap 1+ swap ;

2.
: exch ( d d' --)
2dup @ swap @ rot ! swap ! ;

3. : r@ r> dup >r ;
```

4.
```
: inc2 ( a b -- a+1 b+1)
1+ swap 1+ swap ;
: dec2 ( a b -- a-1 b-1) \ для удовлетворения симметрии
1- swap 1- swap ;

: cp1 ( o d -- o+1 d+1)
2dup swap c@ swap c! inc2 ;
```

5.
```
: cp* ( o d n -- )
0 do cp1 loop drop drop ;
```

6. Одна версия для символов и другая для ячеек.

```
  : 2@+1 ( d d’ -- d+1 d’+1 v v’)
2dup inc2 2swap c@ swap c@ swap ;

: 2@+c ( d d’ -- d+c d’+c v v’)
2dup 1 cells + swap 1 cells + swap
2swap @ swap @ swap ;
```

7. Для перемещения блоков памяти мы напишем слова mov+ и mov- : up и down. Поскольку конечная позиция может перекрываться с начальной позицией, при движении вверх байты будут перемещены из последнего блока назад к базовому адресу, который будет перемещаться последним байтом. При сдвиге вниз он будет продолжаться путем сдвига от первого байта блока к последнему. Наблюдать что у нас уже есть слово cp*, но «оно не учитывает возможные закрылки.

```
: mov+ ( d n p --)
>r tuck 1- + r> rot
0 do
      2dup over +
      swap c@ swap c!
      swap 1- swap
loop drop drop ;

: mov- ( d n p --)
swap >r
over swap -
r> 0 do
      cp1
loop drop drop ;
```

Обратите внимание, что в mov- первые две строки касаются только подготовки адреса источника и адреса назначения. Эти адреса используется словом, которое у нас уже было: cp1, которое делает копию байт и обновляет, увеличивая их, два адреса. Видно, что, по сравнению с ним mov+ заметно сложнее и может можно было бы переписать, если бы у нас было слово, аналогичное cp1 , которое просто уменьшало бы адреса, а не увеличивало их. Но это Более того, если бы мы это сделали, мы могли бы реализовать две версии cp * и на его основе создать две версии mov. Сейчас самое время выделить одну из характеристик программирования на Форте, то есть хорошую программирование на Форте. И это так: фактор, когда это возможно. Мы не Написать mov+ было сложно , но мы поняли, что это возможно. лучше и что для этого нам нужно другое слово, которое, не специфичное для нашей проблемы, могло бы использоваться в будущем для факторизации другие слова.

А в Форте сложное слово можно разложить, факторизовать на более простые слова, и эти слова, если они достаточно
Все просто: их можно отделить от исходной задачи и преобразовать в части, пригодные для использования в различных задачах.
Поэтому: мы никогда не должны упускать возможность сделать вещи лучше, поскольку мало того, что они работают, их еще нужно написать правильно, то есть максимально близко к идеалу. Наша неопытность или лень не меняют этого правила.

1 .
Вот результат пересмотра некоторых уже написанных слов в свете проблемы, поставленной в этом упражнении:

```
: cp1+ ( o d -- o+1 d+1)
2dup swap c@ swap c! inc2 ;

: cp1- ( o d -- o+1 d+1)
2dup swap c@ swap c! dec2 ;

: cp*+ ( o d n -- )
0 do cp1+ loop drop drop ;

: cp*- ( o d n -- )
0 do cp1- loop drop drop ;

: mov+ ( d n p --)
>r tuck + 1- dup r> + rot cp*- ;

: mov- ( d n p --)
swap >r over swap - r> cp*+ ;
```

23. Целые числа

1.

```
: max-cd ( a b -- mcd(a,b))
      ?dup if
            tuck mod recurse
      then ;
```

2. Это превосходный пример, демонстрирующий лаконичность Форта. Программист, думающий на C или переводящий код C в код Forth, может небрежно написать:

```
 : primos? ( a b -- f)
max-cd dup 1 = if drop -1 else drop 0 then ;
```

Но если вы подумаете об этом, вы можете заметить, что max-cd уже оставляет результат в стеке, и воспользоваться этим результатом:

---

*1Разумеется, это правило не меняется из-за каких-либо ошибок или недостатков кода, которые я обнаружил в этой книге.

```
: primos? ( a b -- f)
max-cd 1 = if -1 else 0 then ;
```

Наконец, в третий момент вы можете осознать, что нужный вам флаг уже получен в результате сравнения перед условным выражением. Теперь это Форт. Раньше это был C:

```
 : primos? ( a b -- f)
      max-cd 1 = ;
```

3.

```
      : min-div ( a -- b)
      2
      begin
            2dup mod 0<>
      while
            1 +
      repeat
      nip ;
```

4. Число является простым, если его минимальный делитель один и тот же:

```
   : primo? ( n -- f) dup min-div = ;
```

5. Поскольку простые числа нечетные, оставим нечетное число в стеке в качестве нижнего предела, что позволит увеличить его на два, ускорив поиск. Таким образом, слово ожидает в стеке верхнего предела и нечетного нижнего предела. Версия:

```
: listar-primos ( a b --)
  cr
  begin
      2dup >
  while
      dup primo? if dup . cr then
      2 +
  repeat drop drop ;
```

и еще один

```
: listar-primos ( a b --)
  do
      I dup primo? if . cr else drop then
      2
      +loop ;
```

6. Итерационная версия:

```
: factor ( n -- )
  cr
  begin
      dup min-div dup . cr 2dup <>
  while
      /
  repeat ;
```

Если x — это число, которое мы хотим факторизовать, а y — его минимальный делитель, то мы печатаем y и применяем факторизацию к x/y. То есть мы можем написать рекурсивную версию:

```
: f0 dup primo?
  if
      . cr
  else
      dup min-div dup . cr / recurse
  then ;

: factor cr f0 ;
```

7. Предположим, что n > 1 и b > 1. Используем функцию , возведенную в **:

```
 : ** ( a b -- a^b ) dup 0=
  if
      drop drop 1
  else
      over swap 1- recurse *
  then ;

: n7 ( n b -- p )
  0
  begin
      1+
      3dup ** <
  until
  nip nip 1- ;
```

8. Предположим, что сначала в стеке найден индекс последнего члена суммы, а затем адрес функции. Частичная сумма и счетчик цикла вводятся с помощью -rot .

```
: n8 0 -rot 0 -rot
  swap 1+ 0 do
  2dup execute
  >r rot r> +
  -rot swap 1+ swap
loop drop drop ;
```

9.

```
: reduce ( a b -- a' b') 2dup max-cd tuck / >r / r>;
: qrot ( p q r -- q r p) >r >r 2swap r> r> 2swap ;
: -qrot ( p q r -- r p q) 2swap >r >r 2swap r> r> ;
: qnip ( p q -- q) rot drop rot drop ;
: qtuck ( p q -- q p q) 2swap 2over ;
```

11.

```
: q+ ( p q -- p+q) rot 2dup * >r rot * -rot * + r> reduce ;
: q- ( p q -- p-q) swap negate swap q+ ;
: q* ( p q -- p*q) rot * >r * r> reduce ;
: q/ ( p q -- p/q) >r * swap r> * swap reduce ;
```

12. Проблема здесь в том, что из-за быстрого роста функции факториала числитель может быстро выйти за пределы диапазона целых чисел. Нет смысла сначала вычислять n! а затем разделите на (n k)!.

$$C_{n,k} = \frac{n(n-1)...(n-k+1)}{k!} \qquad (2.1)$$

Таким образом, мы вычисляем выражение как
н(н 1)...(н к + 1)
!

Тогда задача сводится к вычислению производителя формы

$$\prod_{j=0}^{p}(n-j) \qquad (2.2)$$

```
: factorial ( n -- n!) dup 0=
  if
     drop 1
  else
     dup 1- recurse *
  then ;

: prod ( n k -- n(n-1)...(n-k))
  1 -rot 0 do tuck * swap 1- loop drop ;

: C(n,k) ( n k -- C(n,k))
  dup factorial -rot prod swap / ;
```

Эта реализация не рассматривает тривиальный случай Cn,0 = 1.

13. Единственный вопрос здесь может заключаться в том, где взять семя. Здесь мы будем использовать последние две цифры указателя (плюс 1, чтобы предотвратить случай, когда мы получим 0). Учитывая начальное число, при каждом вызове rnd есть новое значение и копия, которую можно использовать.

```
: rnd-seed ( -- a) here 100 mod 1+ ;
: rnd ( a -- b b) 899 * 32768 mod dup ;
```

14.

```
: nsqrt ( a -- a nsqrt(a))
  0
  begin
     1+ 2dup dup * <=
  until 1- ;
```

15. Слово nsqrt оставляет pyn в стеке в следующем порядке. Разделив каждое из них на 1, мы получим рациональное представление. Вам останется лишь провести указанные рациональные операции:

```
: qsqrt nsqrt 1 tuck qtuck q/ q+ 1 2 q* ;
```

16. Первая версия проста и печатает двоичное представление, начиная с младшего бита. Вторая версия сначала вычисляет биты, сохраняет их в стеке, а затем печатает их в обратном порядке, начиная со старшего бита.

```
: .b-> ( n --)
  begin
      dup 2 mod 48 + emit 2 / dup 0=
  until drop ;

: .b ( n -- ) 0 swap
  begin
      dup 2 mod -rot swap 1+ swap 2 / dup 0=
  until drop
0 do
      48 + emit
loop ;
```

17.

```
: .h ( n -- ) 0 swap
  begin
      dup 16 mod -rot swap 1+ swap 16 / dup 0=
  until drop
  0 do
      dup 9 > if 55 + emit else 48 + emit then
  loop ;
```

18. `: bit? ( n b -- f) 0 ?do 2 / loop 2 mod ;`

19.

```
: bit-a-1 ( n b -- n’) 1 swap lshift or ;
: bit-a-0 ( n b -- n’)
  2dup bit? 1= if
      1 swap lshift xor
  then ;
```

20.

```
: c+ ( p q -- p+q) rot + -rot + swap ;
: c- ( p q -- p-q) rot swap - -rot - swap ;
: c*imag ( p q -- a) >r * swap r> * + ;
: c*real ( p q -- a) rot * -rot * swap - ;
: c* ( p q -- p*q) 2over 2over c*imag >r c*real r> ;
: c**( p n -- p^n) 1- 0 do 2dup c* loop ;
```

21.

```
: c/real ( p q -- a) rot * -rot * + ;
: c/imag ( p q -- a) >r * swap r> * - ;
: c/ ( p q -- p/q)
  2dup
  dup * swap dup * + >r \ denominador
  2over 2over
  c/imag >r c/real r> r> ;
```

## 2.4. Векторы

Будем обозначать заглавными буквами «он делает» от слова вектор).

1.

```
: vector create dup , cells allot
```

```
does> dup @ swap cell+ swap ;
```

2. Первая версия удаляет размеры вектора из стека, добавляет их к адресу, с которого начинается смещение, и выполняет присваивание:

```
: v!a ( v i d N -- ) drop swap cells + ! ;
```

Вторая версия может использовать размер вектора, чтобы гарантировать, что он не выйдет за пределы:

```
: v! ( v i d N -- ) rot min cells + ! ;
```

3. Как и в предыдущем упражнении, убеждаемся, что мы не хотим читать позиции за пределами вектора:

```
: v@ ( i d N -- v) rot min cells + @ ;
```

4.

```
: v. ( d N -- )
  cr 0 do
      dup @ . cr cell+
  loop drop ;

: v(.) ( a b d N -- )
  drop rot tuck cells +
  -rot swap cr do
      dup @ . cr cell+
  loop drop ;
```

5. Эта версия также безопасна: в ней хранится не более элементы, которые может содержать вектор.

```
: v!! ( a1...aj m d N --)
  rot min 0 do
  tuck ! cell+
  loop drop ;
```

6.

```
: v-sum ( d N -- suma)
  0 -rot 0 do
      tuck @ + swap cell+
  loop drop ;
```

7.

```
: v-max ( d N -- v )
  over @ -rot 1 do
      cell+ tuck @ max swap
  loop drop ;

: v-min ( d N -- v )
  over @ -rot 1 do
      cell+ tuck @ min swap
  loop drop ;
```

8.

```
: v-pos ( v d N -- i)
  drop 0
  begin
      2dup cells + @
      swap 1+ swap
      >r rot dup >r -rot
      r> r> =
  until nip nip 1- ;
```

9. `: v-med ( d N -- a) 2dup v-sum swap / nip ;`

10. 
```
: 0v!! ( d N -- ) 0 do dup 0 swap ! cell+ loop drop ;
```

11. 
```
: nv!! ( a d N -- ) 0 do 2dup ! cell+ loop drop drop ;
```

12.
```
: v-map ( xt d N -- )
  0 do
      2dup @ swap execute
      over ! cell+
  loop drop drop ;
```

13.
```
: v-map-i ( xt d N -- )
  0 swap 0 do
      3dup tuck cells +
      -rot swap execute
      swap ! 1+
  loop drop drop drop ;
```

14. Гроблема с этой функцией: если она основана на словах rnd-seed и rnd, поскольку вторая использует первую, то если вы хотите заполнить несколько векторов случайными значениями, последовательность из n' чисел, которые
полученное будет одинаково для всех, пока словарь остаётся неизменным, ведь как мы говорили, rnd-seed берёт отсюда две последние цифры . Вот почему мы предпочитаем вручную подавать по одному семени за раз.

```
: v-rnd ( a d N --)
  0 swap 0 do
      3dup cells + !
      1+ rot rnd drop -rot
  loop drop drop drop ;
```

15. Естественно создать цикл и прервать его, если найдено совпадение между предоставленным значением и одним из элементов вектора. С помощью выхода из цикла вы можете прервать цикл и выйти из функции. В этом случае в стек помещается -1 (истина), а цикл и функция прекращаются. Если цикл завершается, в стек помещается 0 (лсжь). В любом случае базовый адрес и искомое значение удаляются из стека, остаются только индекс и флаг. Это вариант решенного ранее упражнения, в котором мы реализуем слово v-pos, предполагающее, что искомое значение действительно находится в векторе.

```
: v-? ( v d N -- f)
  0 swap 0 do
      3dup cells + @
      = if
            nip nip -1 unloop exit
      then
      1+
  loop nip nip 0 ;
```

16.
```
: exch ( d e --) 2dup @ swap @ rot ! swap ! ;
: v-exch ( a b d N --)
  drop tuck swap cells +
  -rot swap cells + exch ;
```

17. Пузырьковый алгоритм — это самый простой из возможных алгоритмов национального упорядочения.
Он берет первый элемент вектора и сравнивает его со вторым и последующими, меняя их каждый раз, когда находится элемент, меньший первого. По завершении первым элементом вектора будет самый маленький из всех. Далее берётся второй элемент и сравнивается с третьим и последующими элементами, производя замену каждый раз, когда находится элемент, меньший второго.

Эта процедура повторяется от первого элемента до предпоследнего. Го завершении вектор полностью упорядочен. Го сути, у нас есть два цикла: один проходит по элементам от первого до предпоследнего;
другой, который для каждого элемента берет следующий и последующие элементы до последнего, производит сравнения и, если необходимо, обмены.

Мы назовем этот второй цикл internal и разделим наш алгоритм на два слова: одно, которое выполняет внутренний цикл, и другое, которое выполняет внешний цикл. Внутренний цикл v-sort-in ожидает в стеке индекса, базового адреса вектора и количества элементов.
В свою очередь, это слово состоит из двух отдельных частей: установки стека в соответствующее состояние для выполнения цикла и выполнения самого цикла. Что касается подготовки стека, эта часть будет иметь эффект ( id N -- d+i d+i+1 i+i N) , где необходимо интерпретировать, что d+i — это адрес индексного элемента i.

```
: v-sort-in ( i d N --)
  >r over >r
  swap cells + dup cell+
  r> 1+ r> swap
  do
      2dup
      @ swap @
      < if
            2dup exch
      then
  cell+
  loop drop drop ;
```

Внешний цикл выполняется немедленно:

```
 : v-sort ( d N --)
   0 -rot dup 1- 0 do
      3dup v-sort-in
      rot 1+ -rot
   loop drop drop drop ;
```

Гримечание: алгоритм пузырька, как уже было сказано, является самым простым из алгоритмов сортировки. В то же время он самый медленный, так как стоимость которого пропорциональна квадрату количества элементов. Поскольку алгоритмы не учитывают, что такое эта константа пропорциональности, иногда случается, что теоретически несовершенные алгоритмы тем не менее практичны. Это один из таких случаев. На нашей машине, ноутбуке Thinkpad с 512 МБ ОЗУ и процессором 1 ГГц, работающем под управлением SuSe Linux, сортировка вектора из 1000 случайных целых чисел занимает незначительное время. Вектор из 4000 случайных целых чисел порядка секунды. Вектор из 8000 случайных целых чисел длительностью порядка трёх секунд. Это означает, что для практических целей этого почти всегда может быть достаточно.

18.

```
: v-invert ( d N --)
  over swap 1 - cells +
  begin
      2dup exch
      1 cells - swap
      1 cells + swap
      2dup >=
  until drop drop ;
```

19.

```
: v-acum ( d N -- )
  over @ swap 1 do
      swap cell+ dup @
      rot + swap 2dup ! swap
  loop drop drop ;
```

2.5. Цепочки

Мы создадим строку как структуру, содержащую две ячейки и саму строку. Первая ячейка указывает максимальную длину строки. Во-вторых, фактически используемая длина. Таким образом, полное описание строки в стеке будет состоять из трех величин: длины, эффективной длины и адреса первого символа. Теперь трех величин много, потому что не все три будут использоваться во всех случаях, и потому что, когда имеется более одного строкового аргумента (например, для объединения строк), в стеке будет шесть чисел, и с ним будет сложно иметь дело. с ними.
По этой причине в разделе do> остается только адрес первого символа строки, а также есть специальные слова для получения максимальной эффективной длины и установки новой эффективной длины. Все строковые функции будут обозначаться символом c в имени. В комментариях стека d указывает направление, максимальную длину N и эффективную длину n .

1.
```
: cadena create dup , 0 , allot
      does> 2 cells + ;
: c-l@ ( d -- N)
      2 cells - @ ;
: c-le@ ( d --n)
      1 cells - @ ;
: c-le! ( n d --)
      1 cells - ! ;
: c-le+1 ( d --)
      dup c-le@ 1+ swap c-le! ;
: c-le-1 ( d --)
      dup c-le@ 1- swap c-le! ;
```

2.
```
: c-type ( d --)
      dup c-le@ 0 do
      @+1 swap emit
   loop drop ;
```

3. Accept — одно из слов, доступных в любой среде Форта. Ожидает в стеке адрес и максимальное количество символов, которые будут прочитаны. Оставляет в стеке количество фактически прочитанных символов. Мы можем использовать его, подготавливая стек перед его вызовом и впоследствии обновляя эффективную длину строки.

```
: c-accept ( d --)
  dup dup c-l@ accept
  swap c-le! ;
```

Мы завершим упражнение, написав нашу собственную версию Accept, которая ожидает в стеке адреса и максимального количества символов для чтения и оставляет общее количество прочитанных символов в стеке. Конечно, простая версия:

```
: intro? 13 = ;

: accept ( d n -- m)
  0 -rot 0 do
      key dup emit
      dup intro? if
            drop drop unloop exit
      else
            over c!
            1+ swap 1+ swap
      then
  loop drop ;
```

4. Операция копирования включает обновление ячейки эффективной длины строки и фактической копии строки.

```
: c-copia ( d d’ --)
```

```
      2dup over c-le@ cp*
      swap c-le@ swap c-le! ;
```

5. Конкатенация возможна, если общая длина строки назначения больше или равна сумме эффективных длин источника и назначения. Слово c-конкат? оставляет в стеке флаг, указывающий, возможна конкатенация или нет. Слово c-concat копирует исходную строку, начиная с последнего символа целевой строки, и оставляет флаг в зависимости от того, можно ли выполнить операцию или нет. Также обновляется эффективная длина строки назначения.

```
: c-concat? ( d d’ --f)
  dup c-l@ -rot
  c-le@ swap c-le@ + >= ;

: c-concat ( o d -- f)
  2dup c-concat? if
      2dup c-le@ swap c-le@ + -rot
      tuck
      dup c-le@ +
      over c-le@ cp*
      c-le!
      -1
  else
      drop drop 0
  then ;
```

6. Две строки равны, если их эффективная длина одинакова и если они соответствуют каждому из своих символов. Поэтому сначала мы напишем слово, которое сравнивает эффективные длины и указывает, равны они или нет. Сравнение строк можно рассматривать как частный случай более общего слова, которое берет из стека два адреса и целое число и сравнивает это количество последовательных позиций. с заданных направлений. Наконец, искомое слово можно записать, объединив первое, которое сравнивает эффективную длину, и второе, присваивая ему начальные адреса и эффективную длину.

```
: c-cmp-le ( d d’ --)
c-le@ swap c-le@ = ;

: c-cmp-n ( d d’ n --f)
  0 do
      2@+1 <> if
            2drop 0 unloop exit
    then
  loop 2drop -1 ;

: c-cmp0 ( d d’ -- f) \ сравнить в «сыром»
      dup c-le@ c-cmp-n ;
: c-cmp ( d d’ --f)
      2dup c-cmp-le if
            dup c-le@ c-cmp-n
      else
            2drop 0
      then ;
```

7. Чтобы указать интервал для сравнения, вы можете указать начальную и конечную позиции или начальную позицию и количество сравниваемых символов. Мы воспользуемся вторым вариантом. На самом деле, представленные нами два слова являются не чем иным, как удобными обертками c-cmp-n , где вместо прямого указания начальных адресов указываются начальные адреса строк и смещение (в первом случае) или смещения (во втором случае). ) из этих источников, а также количество символов для сравнения.

```
: c-cmp-org= ( d d’ n m --)
      >r tuck + -rot + r> c-cmp-n ;
```

Как только это будет сделано, вы сможете почти бесплатно сравнить одинаковое количество символов, но из разных позиций в двух строках. Если a и b — эти позиции, a n — количество символов для сравнения:

```
: c-cmp-org<> ( d d’ a b n --f)
      >r >r rot + swap r> + r> c-cmp-n ;
```

8. Если родительская строка имеет длину l0 , а подстрока имеет длину l1, то, по сути, речь идет о сравнении l1 символов, всегда принимая за начало координат в подстроке адрес первого символа, а в строке - последовательные позиции от первого символа. не более чем до символа l0 l1.

```
: c-subc-f ( d d’ -- f)
  2dup c-le@ swap c-le@ swap
  tuck - 1+ 0 do
  3dup c-cmp-n if
      drop drop drop
      -1 unloop exit
  else
      rot 1+ -rot
  then
  loop
  drop drop drop 0 ;
```

Теперь мы напишем вторую версию предыдущего слова, которая вместо возврата флага возвращает -1, если подстрока не содержится в строке или в позиции внутри строки, из которой находится подстрока. В предыдущем слове базовый адрес внутри цепочки обновляется внутри цикла. Если подстрока найдена, просто вычтите текущий базовый адрес внутри строки из начального адреса строки, чтобы получить позицию подстроки. Поэтому эта версия основана на том, что в первую очередь сохраняется копия начального адреса цепочки. Независимо от того, будет ли сравнение успешным или нет, все остальное, по сути, создает стек для вывода.

```
: c-subc-p ( d d’ -- f)
  over swap
  2dup c-le@ swap c-le@ swap
  tuck - 1+ 0 do
      3dup c-cmp-n if
      drop drop swap -
   unloop exit
  else
      rot 1+ -rot
  then
loop
drop drop drop drop -1 ;
```

9. Очевидно, что необходимы два слова: одно для проверки того, что символ является строчной буквой, а другое — для проверки того, что это заглавная буква. Мы пользуемся этой возможностью, чтобы написать еще един, который сообщает, является ли символ пробелом или нет:

```
: ascii-min? dup [char] a >= swap [char] z <= and ;

: ascii-max? dup [char] A >= swap [char] Z <= and ;

: c-min ( d --)
  dup c-le@ 0 do
      dup c@ ascii-max? if
            dup c@ 32 + over c!
      then
      1+
loop ;

: c-may ( d --)
  dup c-le@ 0 do
```

```
      dup c@ ascii-min? if
           dup c@ 32 - over c!
      then
  1+
loop ;
```

10. Это упражнение — всего лишь частный случай, который можно реализовать с помощью c-subc-pos, когда подстрока имеет длину 1. Однако мы пишем конкретный вариант.

```
: c-char-pos ( d c -- n)
  over swap
  over c-le@ 0 do
      2dup swap c@ = if
       drop swap -
   unloop exit
   else
      swap 1+ swap
   then
loop drop drop -1 ;
```

11. Удаление символа из строки включает в себя две операции: перемещение назад на одну позицию всех символов, следующих за тем, который вы хотите удалить, и изменение эффективной длины строки. У нас уже есть слово, которое перемещает блоки памяти, и другое, изменяющее эффективную длину, поэтому просто объедините их. Если базовый адрес строки равен d, а p — позиция удаляемого символа, аргументами mov- являются d+p+1, l p 1 и 1, где l — эффективная длина цепочки. .

```
: c-x ( p d --)
  tuck 2dup + 1+ -rot
  c-le@ swap - 1-
  1 mov-
  c-le-1 ;
```

12. Теперь мы должны сначала переместить часть строки длины l из d + p на одну позицию, обновить эффективную длину и, наконец, вставить символ:

```
: c-X ( p d --)
  2dup c-le@ 1- < if
      c-x
  else
      nip c-le-1
  then ;

: c-a ( c d --)
      tuck dup c-le@ + c!
c-le+1 ;
```

Легко проверить, что последние два слова не являются робастными. cx завершится неудачно, если вы хотите удалить последний символ строки, а ci завершится неудачей, если вы хотите вставить символ после последнего. Это классический урок отладки: ошибки часто прячутся в углах. В данном случае на границах цепи. Алгоритм, который мы реализовали, предполагает, что мы собираемся действовать в промежуточной позиции, и терпит неудачу, если это не так. Можно возразить, что insert не имеет смысла в отношении позиции за последним символом, но ясно, что должна быть возможность удалить последний символ.
В этих случаях есть три решения: либо переосмыслить алгоритм, чтобы он работал в любом случае, либо инкапсулировать слово, которое не работает
.

---

*2См., например, «Практика программирования» Б. Кернигама и Р. Пайка.

в другом, который выполняет проверки, или прийти к выводу, что случай, который не удался, на самом деле является операцией другого характера (это случай между вставкой символа в строку и добавлением символа в строку). Априори непросто

выбрать, по какому пути идти, поскольку алгоритм, работающий в любом случае: а) может быть трудно найти, б) может иметь некомпенсирующую сложность и в) может требовать модификационных слов, которые не должны быть изменены, потому что это общие слова, которые уже хорошо подходят для того, для чего они были написаны. Но, конечно, когда этот алгоритм существует и пригоден, ему следует отдать предпочтение.

В данном случае решение такое: что касается cx, то мы инкапсулируем его в cX , который проверяет, является ли исключаемая позиция последней в цепочке, и в этом случае ограничивается уменьшением эффективной длины на одну единицу. цепь; По поводу ci приходим к выводу, что вставка символа — это операция иного характера, чем добавление символа сверх предыдущего, поэтому реализуем слово
c-a

```
: c-X ( p d --)
  2dup c-le@ 1- < if
      c-x
  else
      nip c-le-1
  then ;

: c-a ( c d --)
      tuck dup c-le@ + c!
c-le+1 ;
```

13. Речь идет о поиске позиции первого символа в строке, отличного от пробела. Если p является этой позицией, это просто вопрос перемещения p позиций влево на блок из l - p байтов, где l - эффективная длина строки, и обновления указанной эффективной длины al-p. Слово c-1 возвращает позицию первого символа без пробелов (ascii 32). Мы предполагаем, что строка не пуста.

```
: c-1 ( d --p) 0 begin 2dup + c@ 32 = while 1+ repeat nip ;

: c-no-blancos< ( d --)
  dup c-1
  2dup 2dup
  + -rot swap c-le@ swap -
  rot dup >r mov- r>
  over c-le@ swap -
  swap c-le! ;
```

14 Речь идет о поиске «последнего» символа в строке, кроме пробела, и настройке эффективной длины. Если позиция этого последнего символа равна p, новая эффективная длина равна l p. Слово c-$ возвращает позицию последнего символа, не являющегося пробелом. Опять же, мы предполагаем, что строка не пуста.

```
: c-$ ( d --p)
  dup c-le@ 1-
  begin 2dup + c@ 32 = while 1- repeat nip ;

: c-no-blancos> ( d--)
  dup c-$ 1+ swap c-le! ;
```

Может быть интересно узнать длину строки от первого символа, не являющегося пробелом, до последнего символа, не являющегося пробелом. Мы можем назвать эту длину кажущейся длиной, поскольку пробелы не видны. Это легко реализовать:

```
: c-la ( d --n)
  dup c-$ swap c-1 - 1+ ;
```

15. Процедура, которой мы будем следовать, будет следующей: мы возьмем начальный адрес цепочки и следующий адрес и будем увеличивать оба адреса, пока второй не достигнет последнего символа цепочки. Если в первом адресе обнаруживается пробел, а во втором — символ, отличный от пробела, то этот второй адрес указывает на начало слова.

Затем адрес сохраняется в стеке, и счетчик также в стеке увеличивается, указывая количество адресов, находящихся в стеке на данный момент. Таким образом, структура стека следующая: один или несколько адресов, число, указывающее количество этих адресов, указатель на место в цепочке и указатель на следующую позицию. В итоге в стеке остается серия адресов и их количество.
Эти значения можно перевести в вектор с помощью v!!. Необходимо рассмотреть случай, когда первый символ строки отличается от пробела, и в этом случае он является началом слова.

```
: c-palabras ( d -- a b c ... n)
  dup c@ 32 <> if
      1 over
  else
      0 swap
  then
  dup 1+
  over c-le@ 1- 0 do
      2@+1
      32 <> swap 32 = and
   if >r swap 1+ over r> then
loop drop drop ;
```

С другой стороны, размещение всех адресов в стеке может быть нежелательным в определенных ситуациях, поскольку другие элементы стека будут недоступны. Тогда была бы полезна другая версия, которая по заданному адресу получала адрес следующего слова. Второстепенная проблема заключается в том, что если вы хотите исследовать цепочку целиком, поскольку конечной метки нет, то на чем остановить поиск? Мы напишем два новых слова: c-#p дает количество слов в строке; c-ps при заданном адресе предлагает адрес следующего слова. Таким образом, первый из них можно использовать для создания цикла, внутри которого одно слово будет последовательно передаваться другому. Эта реализация c-ps , которая требует только одного адреса и возвращает адрес, завершается сбоем, если первый или последний символ в строке не является пробелом. Но мы предпочитаем чистое использование создаваемого стека, поэтому, когда необходимо проанализировать строку, мы принимаем меры предосторожности, вставляя в начале и добавляя в конце пробел. Ибо когда это необходимо, мы пишем c-ребра

```
: c-#p ( d --n)
  dup c@ 32 = if
      0 swap
  else
      1 swap
  then \ c d
  dup c-le@ 1- >r
  dup 1+ r> 0 do \ c d d+1
  2@+1 32 <> swap 32 = and if
      rot 1+ -rot
  then
loop 2drop ;

: c-saltar-blancos ( d --d’)
  begin
      dup c@ 32 =
  while
      1+
  repeat ;

: c-saltar-no-blancos ( d --d’)
  begin
      dup c@ 32 <>
  while
      1+
  repeat ;

: c-bordes ( d -- d)
```

```
  dup 32 swap 0 swap c-i
  dup 32 swap c-a ;

: c-ps ( d --d')
  saltar-no-blancos
  saltar-blancos ;
```

16.

```
: c-vacia? ( d --f)
  dup c-le@ 0 do
      @+1 32 <> if
            drop 0 unloop exit
      then
loop drop -1 ;
```

17. Идея состоит в том, чтобы взять базовый адрес строки и следующий адрес и увеличивать их, пока второй не достигнет конца строки. Если содержимое обоих адресов пусто, то символ, на который указывает второй, удаляется с помощью cx.

Поскольку эффективная длина строки изменяется по мере удаления символов, мы не можем пройти по ней с помощью цикла do....

```
: c-purgar ( d --)
  0 begin
      2dup swap c-le@ 2 - <=
  while
      2dup + dup 1+
      c@ 32 = swap
      c@ 32 = and if
            2dup 1+ swap c-x
      else
            1+
      then
  repeat 2drop ;
```

Проблема здесь в том, что необходимо контролировать, чтобы оно не исследовалось дальше конца цепочки. Вот почему полезно слово, которое по индексу и базовому адресу указывает, указывает ли индекс на какую-то позицию внутри цепочки.

```
: c-dentro? ( n d --f)
  c-le@ < ;
: c-lpn ( n d --m)
  0 -rot begin
      2dup c-dentro? >r
      2dup + c@ 32 <> r> and
  while
      rot 1 -rot
      swap 1+ swap
  repeat drop drop ;
```

Эта версия не подходит для использования вместе с c-словами, так как оставляет в стеке абсолютные адреса, а не пары (смещение, базовый адрес). По этой причине мы напишем другую версию, которая принимает только один адрес и возвращает количество символов без пробелов, следующих за этим адресом. Одна из проблем заключается в том, что в строке отсутствует конечный знак, и поэтому слово завершится ошибкой, если последний символ в строке не является пробелом.

```
: c-lp ( d –n) 0 begin over c@ 32 ¡¿while 1+ swap 1+ swap repeat nip ;
```

19. Скорректируем влево. Слово c-1 указывает количество пробелов, которые необходимо устранить. Поскольку эффективная длина строки должна быть сохранена, для каждого удаленного пробела в начале строки в конец строки необходимо добавить еще один пробел.

```
: c-< ( d --)
```

```
  dup c-1 0 do
      dup 0 swap c-x
      dup 32 swap 0 swap c-i
  loop drop ;
```

Чтобы переместить вправо, необходимо удалить конечные пробелы и вставить их в начало.

```
: c-> ( d --)
  dup dup c-le@ swap c-$ - 1- ?do
      dup dup c-le@ + 1- c-x
      dup 32 swap 0 swap c-i
loop drop ;
```

Для настройки на центр полезна концепция видимой длины, которую мы определили и реализовали с помощью c-la:

```
: c->< ( d --)
  dup c-<
  dup dup c-le@ swap c-la
  - 2 / 0 ?do
      dup dup c-le@ 1- swap c-x
      dup 32 swap 0 swap c-i
loop drop ;
```

Эти слова удобно печатать столбцами. Учитывая строку, мы можем дополнять ее пробелами до тех пор, пока она не достигнет определенной длины, а затем обернуть ее влево, вправо или по центру перед печатью.

2.6 Матрицы

1. Помимо знания количества строк и столбцов, интересно знать общее количество элементов матрицы, поскольку, поскольку ее элементы располагаются в памяти линейно, возникнут ситуации, в которых ее удобно рассматривать как вектор (например, чтобы найти
максимальное и минимальное значения и другие подобные значения). Вот почему мы вводим mN , который принимает адрес и возвращает тот же адрес и количество элементов матрицы, оставляя стек пригодным для использования уже разработанных векторных функций.

```
: matriz create 2dup swap , , * cells allot
      does> 2 cells + ;

: m-f ( d --f)
      2 cells - @ ;

: m-c ( d --f)
      1 cells - @ ;

: m-N ( d -- d N)
      dup m-f over m-c * ;
```

2. Мы включаем механизм, хотя и простой, для предотвращения чтения или записи за пределами размеров массива. Если f — количество строк, а c — количество столбцов матрицы, то смещение в ячейках от начального адреса элемента координат (i, j ) равно i x c + j . Это смещение должно быть меньше f x c Строки и столбцы индексируются от 0. Поскольку, учитывая пару значений строки и столбца и базовый адрес матрицы, нам нужно будет найти адрес данного элемента более одного раза, мы учтем эту операцию в слово m-dir , которое также гарантирует, что оно не выйдет за пределы.

```
: m-ij->l ( i j d -- d’)
  rot over m-c * rot +
  over dup m-c swap m-f * min
  cells + ;
```

```
: m! ( v i j d --)
  m-ij->l ! ;
```

3.

```
: m@ ( i j d -- v)
     m-ij->l @ ;
```

Разделяем впечатление на два слова. Внутренний цикл перебирает элементы строки и печатает cr перед переходом к следующей строке.

```
: m.0 ( i d --)
  0 swap dup m-c 0 do
      3dup m@ 12 .r swap 1+ swap
  loop cr drop drop drop ;

: m. ( d --)
     cr 0 swap dup m-f 0 do
           2dup m.0
           swap 1+ swap
loop drop drop ;
```

5. Это ситуация, когда мы можем рассматривать матрицу как вектор. Эквивалентная функция для векторов у нас уже есть, осталось только получить количество элементов матрицы.

```
: 0m!! ( d --)
  dup m-c over m-f * 0v!! ;
```

6. Как и в предыдущем слове, используем готовый вариант для векторов.

```
: m-map ( xt d -- )
  dup m-c over m-f * v-map ;
```

7. Этот случай немного отличается от предыдущего и мы не можем сделать вызов v-map-i из нашего, так как в версии для векторного выполнения он ожидает один аргумент в стеке, а теперь нам нужно два.
Да, мы можем сохранить точку зрения, что матрица — это вектор, и для каждого индекса получить соответствующую строку и столбец, применив к ним функцию.

```
: m-l->ij ( l d -- i j)
  m-c /mod swap ;
: m-map-ij ( xt d --)
  0 swap dup m-c over m-f * 0 do
  3dup 2dup swap cells + >r
  m-l->ij rot execute
  r> !
  swap 1+ swap
loop drop drop drop ;
```

8. Воспользуемся векторной версией.

```
: m-max ( d -- )
  m-N v-max ;

: m-min ( d -- )
  m-N v-min ;
```

9. Опять воспользуемся векторной версией:

```
: m-rnd ( s d --)
  dup m-c over m-f * v-rnd ;
```

Очевидно, что многие другие функции для матриц могут быть основаны на функциях для векторов, и на этом нет необходимости настаивать, поскольку процедура очевидна.

10. Для этого слова, как и для следующего, будет полезно слово, меняющее местами любые два элемента массива. Он будет ждать в стеке строку и столбец первого элемента, строку и столбец второго элемента и базовый адрес:

```
: m-exch ( i j p q d --)
  dup >r m-ij->l -rot r> m-ij->l exch ;
```

Теперь легко поменять местами две строки:

```
: m-exch-f ( i j d --)
  0 swap dup m-c 0 do
      2over 2over
      >r tuck r> m-exch
      swap 1+ swap
loop
4 0 do drop loop ;
```

или две колонки

```
: m-exch-c ( i j d --)
      0 swap dup m-f 0 do
            2over 2over
            >r dup >r -rot r> swap r> m-exch
            swap 1+ swap
loop
4 0 do drop loop ;
```

11. Чтобы транспонировать матрицу, необходимо пройти по всем ее строкам, кроме последней, и для каждой строки i рассмотреть элементы, найденные в столбцах j > i. Вот почему мы разделяем операцию на два слова, по одному на каждый цикл:

```
: m-'-f ( i d --) \ para la fila i
  over 1+ tuck over m-c swap do
  3dup >r 2dup swap r>
  m-exch
  swap 1+ swap
loop drop drop drop ;

: m-' ( d --)
  0 swap dup m-f 1- 0 do
  2dup m-'-f
  swap 1+ swap
loop drop drop ;
```

12. Умножение строки или столбца на число. Число, строка или столбец и базовый адрес матрицы ожидаются в стеке.

```
: m-f* ( n i d --)
  0 swap dup m-c 0 do
      2over 2over
  m-ij->l dup >r @ * r> !
  swap 1+ swap
loop
  4 0 do drop loop ;

: m-c* ( n j d --)
  0 -rot dup m-f 0 do
      2over 2over
      m-ij->l dup >r @ * r> !
      rot 1+ -rot
  loop
  4 0 do drop loop ;
```

13. Идея состоит в том, чтобы расположиться по начальным адресам каждой строки и продвигаться по одной ячейке столько раз, сколько столбцов в матрице. Слова ожидают в стеке двух строк и базового адреса массива. Первая строка заменяется ее сложением или вычитанием второй, которая остается неизменной.

```
: m-ff'+ ( f f' d --)
  dup m-c >r
  tuck 0 swap m-ij->l -rot 0 swap m-ij->l
  r> 0 do
      2@+c + over 1 cells - !
  loop 2drop ;

: m-ff'- ( f f' d --)
  dup m-c >r
  tuck 0 swap m-ij->l -rot 0 swap m-ij->l
  r> 0 do
      2@+c swap - over 1 cells - !
  loop 2drop ;
```

2.7. Численные вычисления

В этом разделе предполагаются операции с вещественными числами. Слова для работы1 с числами этого типа можно найти в руководстве Gforth, системе, кот

оруюмы1
используем, начиная со страницы1 60 используемого нами издания. В общем,
3
существует параллель между целочисленными словами и реальными словами, например. например cTop для целых чисел и fcfop для реальных. Важным отличием является то, что размер действительных чисел не совпадает с размером целых чисел. Эквивалентное слово для ячеек — float. Другое, что настоящие лечатся в своем стеке, и что f>r нет. Чтобы1 сохранить функциональность >r, мы1 можем поместить настоящее где-нибудь safe , как здесь. Например:

```
: f>r here f! ;
: fr> here f@ ;
```

Мы также могли бы1 реализовать свой собственный стек, но нас устраивает это простое решение. Также необходимо учитывать, что при обнаружении в stack вещественных и целочисленных аргументов , как это происходит в первом упражнении, мы1 фактически имеем дело с двумя стеками. Чтобы1 избежать путаницы1 мы1 разделяем содержимое обоих в комментариях двоеточием. Ниже добавим полезные для нас аналоги реалов:

```
: f2dup fover fover ;
```

---

*3Поскольку этот документ находится в постоянном процессе пересмотра, мы1 не можем гарантировать совпадение. совпадение с другими редакциями или редакциями

```
: -frot frot frot ;
: s>f s>d d>f ;
: f2over ( x y z t -- x y z t x y)
  here f!
  here 1 floats + f!
  f2dup
  here 1 floats + f@ -frot
  here f@ -frot ;

: @+f ( : d -- v : d+float)
  dup f@ 1 floats + ;
```

1.

```
: espacio 32 emit ;
: espacios 0 do espacio loop ;
```

```
: ftabular ( h a b : xt --)
  cr
  begin
      f2dup f<
  while
      fover f. 10 espacios
      fover dup execute f. cr
      -frot fover f+
      frot
  repeat
  drop \ ahora tengo que limpiar
  3 0 do fdrop loop ; \ dos pilas
```

2.

```
: fvector create dup , floats allot
  does> dup @ swap cell+ swap ;
: fv! ( v i d N --)
  rot min floats + f! ;
: fv@ ( i d N --)
  rot min floats + f@ ;
```

3.

```
: ftab ( a h : xt d N --)
  0 do
      fover over execute \ a h v : xt d
      dup f! \ a h : xt d
      ftuck f+ fswap \ a+h h : xt d
      1 floats + \ a+h h : xt d'
  loop drop drop fdrop fdrop ;
```

МЫ можем подготовить таблицу равноотстоящих друг от друга аргументов, чтобы применить к ним некоторуюфункцию Если мы объявим функциюидентификации:

```
: f-identidad ;
```

использование в сочетании с ftab позволяет нам заполнить вектор аргументами. Например, чтобы заполнить вектор из 10 элементов В этом контексте также полезна реальная версия v-map которая применяет функциюк каждому элементу вектора:

```
: fv-map ( : xt d N --)
  0 do
      2dup f@ execute
      dup f!
      1 floats +
  loop drop drop ;
```

Может бьть полезно иметь отдельно два вектора: вектор аргументов и вектор значений, полученных из них применением заданной функции. Для этого бьло бы полезно скопировать один вектор действительных чисел в другой:

```
: fv-copia ( : d N d' N --)
  drop swap floats 1 cells + cp* ;
```

4. Это настоящая версия v!!

```
: fv!! ( a1...aj m d N --)
  rot min 0 do
      dup f! 1 floats +
  loop drop ;
```

Единственный недостаток - значения приходится вводить в обратном порядке. Это не проблема, потому что у нас есть слово v-invert, для которого мы можем написать версиюдля реальных векторов:

```
: fexch ( d d' --)
```

```
  2dup f@ f@ f! f! ;

: fv-invert ( d N --)
  over swap 1 - floats +
  begin
      2dup fexch
      1 floats - swap
      1 floats + swap
      2dup >=
  until drop drop ;
```

5. Следует отметить, что ряд представляет собой сумму общрго слагаемого Tn = x n/n! и что рекуррентное соотношение Tn+1 = xTn/(n+1) существует. Воспользуемся этим обстоятельством для ускорения расчета. С другой стороны! ряд бесконечен, и нам придется вычислять максимальное количество членов. Критерием, который мы примем, будет остановка расчета в термине Tn < 10 9 . Стек будет содержать значение, которое будет увеличиваться на 1 в каждом термине. Что касается стека действительных чисел, он будет содержать частичнуюсумму, аргумент и последний член.

```
: fe^ ( x -- e^x)
  1 \ n
  1.0e fswap \ частичная сумма
  1.0e \ T0
  begin
      fdup 1e-9 f>
  while
      fover f*
      dup s>f f/ 1+
      frot fover f+ -frot
  repeat
  fdrop fdrop drop ;
```

6. В методе пополам берется промежуточное значение m= (a+b)/2 и проверяется, равен ли знак f(m) знаку f(b) или отличен от него.
Если он другой, корень ищется в интервале [m b]. Если оно равно, корень ищется в интервале [a, m]. Считаем, что корень r найден, если f(r) < 10 9 . Рекурсивная реализация следует из определения самого алгоритма. Слово, которое мы собираемся записать, ожидает в реальном стеке конца интервала и в стеке адреса функции, корень которой ишртся. Предположим, что такой корень существует.

```
: fbisec ( a b : xt -- r:)
  fover dup execute \ es 'a' la raiz?
  1e-9 f< if fdrop drop exit then
  fdup dup execute \ es 'b' la raiz?
  1e-9 f< if fnip drop exit then
  f2dup f+ 2.0e f/ \ punto medio 'm'
  dup execute fswap
  dup execute f* \ f(b)*f(m)
  f0< if
      frot fdrop fswap recurse \ busca en [m,b]
  else
      fnip recurse \ busca en [a,m]
  then ;
```

7. Идея здесь проста: разделить интервал на подинтервалы1 для каждого подинтервала проверить, есть ли смена знака функции, и если да, то найти корень в этом подинтервале с помощьюслова fbisec.

```
: hay-raiz? ( a b : xt -- :f)
  dup execute fswap execute
  f* f0<= ;

: fraices ( a b : xt --)
  cr fover f- 1e2 f/ \ a h : xt
```

```
  100 0 do
      f2dup fover f+
      dup hay-raiz? if
            f2dup fover f+ dup
            fbisec f. cr
      then
  fswap fover f+ fswap
loop fdrop fdrop drop ;
```

8. Сначала нам нужен приблизительный корень. После получения мы1 вычисляем последовательные приближения до тех пор, пока разница между двумя последовательными приближениями не станет меньше 10 9 . fsqrt0 вычисляет приблизительный корень своего аргумента. fsqrt1 вычисляет лучший корень на основе аргумента и предыдущего корня.

```
: fsqrt0 ( x: -- r:)
  1.0e begin
  f2dup fdup f* f>
  while
      1.0e f+
  repeat fnip ;

: fsqrt1 ( x r0: -- x r0 r1:)
  f2dup f/ fover f+ 0.5e f* ;

: fsqrtN ( x: -- r:)
  fdup fsqrt0 fsqrt1
  begin
      f2dup f- fabs 1.0e-9 f>
  while
      fnip fsqrt1
  repeat fnip fnip ;
```

9. Слово, которое мы напишем, состоит из двух частей, и их можно разложить на множители. В первой части вектор заполняется значениями функции: f(a), f(a + I)... до f(b). Во втором вычисляется интеграл. В стеке ожидается адрес функции, которуювы хотите интегрировать, адрес вектора и количество его элементов. В куче реалов a и b.
Если вектор состоит из n элементов, то интервал h равен (b a)/(n 1). Специально объявлять вектор не обязательно: отсюда можно использовать пространство .
1.0e начало

```
: fintegral ( a b: xt d N -- i:)
  2dup >r >r rot r> r> \ guardo (d,N)
  fover f- dup 1- s>f f/ \ вычислить h\
  0 do \ заполнить вектор
      fover over execute \ таблицей значений
      dup f! 1 floats +
      ftuck f+ fswap
  loop fnip drop drop \ сохранить 'h'
  0.0e \ аккумулятор s
  1- 0 do \ s:d
      dup f@
      1 floats + dup f@
      f+ 0.5e f* f+
  loop f* drop ;
```

10. Учитывая набор значений (xi , yi),
мы намерены скорректировать их с помошьюлинии
y = ax+b. Оптимальное соответствие достигается для тех значений (a, b),

$$S(a,b) = \sum_i [y_i - (ax_i + b)]^2 \qquad (2.3)$$

при которых расстояние между при котором расстояние между точками и линией минимально. Это формализуется рассмотрением функции

$$\begin{aligned} as_{x^2} + bs_x &= s_{xy} \\ as_x + bN &= s_y \end{aligned} \qquad (2.4)$$

$$a = \frac{Ns_{xy} - s_x s_y}{Ns_{x^2} - (s_x)^2} \qquad (2.5)$$

$$b = \frac{s_{x^2} s_y - s_x s_{xy}}{Ns_{x^2} - (s_x)^2} \qquad (2.6)$$

которая будет минимальной, когда две ее частные производные по a и по b равны нулю
Отсюда следует система двух уравнений, обеспечивающая (а, б):
asx2 + bsx = sxy asx + bN = sy

$$\begin{aligned} s_x &= \sum_i x_i \\ s_{x^2} &= \sum_i x_i^2 \\ s_y &= \sum_i y_i \\ s_{xy} &= \sum_i x_i y_i \end{aligned} \qquad (2.7)$$

Итак, вначале нам нужно написать слова,
которые по заданному вектору стека или тому и другому вычисляют суммы

```
: fv-sum ( :d N -- s:)
  0.0e 0 do
      dup f@ f+
      1 floats +
  loop drop ;

: fv-sum2 ( :d N -- s2: )
  0.0e 0 do
      dup f@ fdup f* f+
  1 floats +
  loop drop ;

: fv-sumxy ( : d N d’ N -- sxy:)
  0.0e drop swap 0 do
      2dup f@ f@ f* f+
      1 floats + swap 1 floats +
  loop drop drop ;
```

и поскольку в выражениях для a и b знаменатель общий (т.е. определитель матрицы1
коэффициентов), запишем также через слово, чтобы1 вычислить его явно:

```
: fv-det ( :d N -- v:)
  2dup fv-sum2 dup s>f f*
  fv-sum fdup f* f- ;

\ определитель \ H
\ сумма xy \ сумма y \ сумма x
```

Теперь мы можем вычислить наклон, предполагая, что сначала в стек помещается вектор x, а затем вектор для и.

```
: fv-pendiente ( : d N d' N -- a:)
  2over fv-det \ determinante
  dup s>f \ N
  2over 2over fv-sumxy \ sum x.y
  fv-sum \ sum y
  fv-sum \ sum x
  f* -frot f* fswap f-
  fswap f/ ;

: fv-ordenada ( : d N d' N -- b:)
  2over fv-det \ determinante
  2over 2over fv-sumxy \ sum x.y
  fv-sum \ sum y
  f2dup fv-sum \ sum x
  fv-sum2 \ sum x.x
  frot f* -frot f* f-
  fswap f/ ;

: fv-ajuste-lineal ( : d N d' N -- a b:)
  2over 2over
  fv-pendiente
  fv-ordenada ;
```

11. Используя определение производной, задача тривиальна:

```
: f' ( x : xt -- y:)
  1e-9 fswap f2dup f+
  dup execute fswap execute f-
  fswap f/ ;
```

12.

```
: fpol-eval ( x: d N -- y:)
  0e fswap
  0 swap 0 do
      2dup fdup
      s>f f** f@ f*
      frot f+ fswap
      1+ swap 1 floats + swap
  loop drop drop fdrop ;
```

Если коэффициенты полинома вводятся с помош^ю^!!, имейте в виду, что значения сохраняется в обратном порядке.

13. Многочлен коэффициентов a0, a1, a2, a3  an имеет производную которая представляется многочленом a1, 2a2, 3a3  0.

```
: fpol-derivada ( : d N --)
  1 swap 1- 0 do
      2dup s>f 1 floats + f@ f*
      over f!
      1+ swap 1 floats + swap
  loop drop 0.0e f! ;
```

14. Устанавливаем константу интегрирования равной 0

```
: fpol-integral ( : d N --)
  2dup 1- floats +
  rot drop swap 1- dup
  0 do
      2dup s>f 1 floats - f@ fswap f/
      over f!
      1- swap 1 floats - swap
```

```
  loop drop 0.0e f! ;
```

15. Используя предыдущее слово, легко найти определенный интеграл от многочлена на отрезке, учитывая, что если D — первообразная числа p(x), то

$$\int_a^b p(x)dx = D(b) - D(a) \qquad (2.8)$$

```
: fpol-integral-definida ( a b:d N--x:)
  2dup fpol-integral
  2dup fpol-eval fswap fpol-eval
  f- ;
```

16. Для предлагаемого уравнения, если h— приращрние времени, x(t + h) рассчитывается на основе x(t) посредством следующих шагов:

```
а)    k1 = f(t)
б)    k2 = f(t + 7)
в)    k3 = f(t + 7 ) г) k4 = f(t + h)
д) x(t + h) = x(t) + ч [к1 + 2(к2 + к3) + к4]
```

$$a)\ k_1 = f(x)$$

$$b)\ k_2 = f(x + \tfrac{k_1}{2})$$

$$c)\ k_3 = f(x + \tfrac{k_2}{2})$$

$$d)\ k_4 = f(x + k_3)$$

$$e)\ x(t+h) = x(t) + \tfrac{h}{6}\left[k_1 + 2(k_2 + k_3) + k_4\right]$$

Напишем слово, ожидакшре в стеках (x0, t0, h) с одной стороны, а с другой стороны! адрес функции f(t) и два вектора для хранения последовательных значений t и x. Сначала с помошью ftab заполняем вектор, содержащей значения времени. После этого из начального значения x мы! получаем последующее и сохраняем их.

```
: frk4-ft ( x0 t0 h : xt d N d' N --)
  f2dup 2swap ['] f-identidad
  -rot ftab \ вектор со значениями t
  0 do
      frot fdup dup f! -frot
      1 floats + \ guardo x
      ftuck \ guardo h
      fover over execute -frot \ k1
      f2dup 0.5e f* f+ over
      execute -frot \ k2
      f2dup f+ over
      execute -frot \ k4
      f+ f>r
      fswap 4.0e f* f+ f+
      fover 6.0e f/ f*
      frot f+ fswap
      fr> fswap
  loop fdrop fdrop fdrop drop drop ;
```

17. Для предложенного уравнения, если h — приращение времени, x(t + h) Он рассчитывается на основе x(t) посредством следующих шагов:

$$a)\ k_1 = f(x)$$
$$b)\ k_2 = f(x + \frac{k_1}{2})$$
$$c)\ k_3 = f(x + \frac{k_2}{2})$$
$$d)\ k_4 = f(x + k_3)$$
$$e)\ x(t+h) = x(t) + \frac{h}{6}\left[k_1 + 2(k_2 + k_3) + k_4\right]$$

Как видно, промежуточные расчеты не требуют t

```
: frk4-fx ( x0 t0 h : xt d N d' N --)
  f2dup 2swap ['] f-identidad
  -rot ftab \ vector con valores de t
  fnip fswap 0 do \ h x : xt d
  fdup dup f! 1 floats +
  fdup over execute fswap \ k1
  f2dup fswap 0.5e f* f+
  over execute fswap \ k2
  f2dup fswap 0.5e f* f+
  over execute fswap \ k3
  f2dup f+
  over execute fswap \ k4
  f>r -frot f+ 2.0e f*
  f+ f+ fover 6.0e f/ f* fr> f+
loop fdrop fdrop drop drop ;
```

Хотя приведенный выше код верен, результаты неприемлемы.
Например, когда f(x) = 1/x, решение уравнения, когда x(0) = 1, равно
$x = \sqrt{2(t + \frac{1}{2})}$
, тогда как численно мы получаем 2), что для t = 0,99 равно x(0,99) = 1,726267. 1,613285. Это иллюстрирует проблемы численного расчета с действительными числами. В этом случае мы знаем аналитическое решение, и поэтому мы смогли обнаружить ошибку, что происходит, когда мы интегрируем уравнение, аналитическое решение которого нам не известно (именно поэтому мы его интегрировали) Используется ли тот же алгоритм? реализованный на C и скомпилированный с использованием gcc, дает правильные результаты как с типом float, так и с типом double.

18. Нам нужен генератор случайных чисел. У нас уже есть генератор целых чисел, из которого мы можем написать еще один настоящий. Как только это будет сделано, реализация станет простой.

```
: frnd
      rnd
      s>f 32768 s>f f/ ;

: fmontecarlo2 ( :xt -- v:)
      0.0e rnd-seed
      1000000 0 do
            frnd frnd over execute f+
      loop 1e6 f/ drop ;
```

2.8. Файлы

\ к3 \к4

1. Основные функции — открытие файла и чтение строки.
Первый ожидает в стеке адреса, содержащего имя файла, длину этого имени и способ его открытия.
Возвращает целое число, идентифицирующее файл для последующих операций, и нулевой флаг.

Возникла какая-то проблема, вместо идентификатора файла возвращает 0 и вместо флага код ошибки.
Что касается строки чтения, то она ожидает в стеке адреса и размера буфера, в котором будет сохранена  строка чтения, и идентификатора файла.
Возвращает количество прочитанных символов и флаг: -1, если все прошло хорошо, и 0 в противном случае.
Он также оставляет в стеке значение 0. При этом трафаретная печать файла требует открытия файла и чтения строк, пока операция возможна. Реализация проста, и здесь мы предполагаем, что ошибок нет
(например, файл не существует или его невозможно открыть).
Следовательно, это слово должно быть инкапсулировано в другое слово, учитывающее эти ошибочные
состояния. list-file ожидает в стеке адреса двух строк (формата, который мы определили в разделе, посвященном строкам): одной, содержащей имя файла, и одной из буфера, в котором будут храниться строки.

```
: f-list ( d d' --)
  swap c-le@ r/o open-file drop \ d' id
  begin
      2dup swap dup c-l@ rot
      read-line drop \ d' id n flag
  while
      >r over r> swap c-le!
      over c-type cr
  repeat
  swap close-file 2drop ;
```

2. Это небольшой вариант

```
: f-list# ( d d' --)
  cr
  swap dup c-le@ r/o
  open-file drop \ d' id
  1 -rot \ cuenta lineas
  begin
      2dup swap dup c-l@ rot
      read-line drop \ c d' id n flag
  while
      >r over r> swap c-le! \ c d' id
      rot dup 6 .r 1+ -rot
      2 espacios over c-type cr
  repeat
swap close-file 3drop ;
```

3. Это слово будет принимать в качестве аргументов имя файла и буфер, куда будут читаться строки.
Опять же, нет обработки ошибок.

```
: f-lee-lineas ( d d' --)
  cr s" C-n nueva linea; C-f fin" type cr
  swap dup c-le@ w/o open-file drop \ d' id
  begin
    key 14 =
  while
      over c-accept
      2dup write-line drop
  repeat 2drop ;
```

4. Этому слову нужны три аргумента: имя файла, имя буфера, в котором будут читаться строки, и вектор, в котором будут храниться числа. Предположим, что пустых строк нет и количество строк в файле соответствует количеству строк в векторе.
Надежная реализация подтвердит, что это так, но это всего лишь упражнение. Нам нужно слово, способное преобразовать строку, представляющую целое число, в целое число с его знаком.

```
: c-a-entero ( d n -- v)
  over c@ [char] - = if
  -1 -rot
  1- swap 1+ swap
else
  1 -rot
then
0 -rot
0 do
  @+1 48 -
  rot 10 * + swap
loop drop * ;

: f-a-vector ( d d’ d’’ N --) \ archivo, buffer y vector
  >r rot dup c-le@ r/o
  open-file drop \ d’ d’’ id
  r> 0 do \ d’ d’’ id
  rot 2dup \ d’’ id d’ id d’
  dup c-l@ \ d’’ id d’ id d’ l
  rot read-line \ d’’ id d’ n flag 0
  drop drop \ d’’ id d’ n
  over c-le! \ d’’ id d’
  dup c-<
  dup dup 0 swap \ d’’ id d’ d’ 0 d’
  c-lpn \ d’’ id d’ d’ l
  c-a-entero \ d’’ id d’ v
  >r rot r> over ! \ id d’ d’’
  1 cells + rot
loop close-file 2drop ;
```

Хотя это довольно длинное слово, оно остается простым, поскольку выражает только линейную последовательность операций. Прежде всего подготовьтесь стек для чтения строки, которая помещается в буфер. Они удалены используя c-< возможные пробелы в начале и длина слова, обозначающего число. Это важно: нет мы можем использовать эффективную длину строки, поскольку она может включать дополнительные пробелы после цифр. После завершения теперь целое число, которое оно представляет, можно вычислить и сохранить в вектор. После увеличения направления вектора в ячейке и Оставив стек таким, каким он был в начале цикла, возвращаемся в начало и читаем следующую строку. После завершения цикла мы закрываем файл и очищаем кучу.

5. Естественно использовать уже имеющиеся у нас c-слова. это слово ожидает в стеке адрес строки и возвращает адреса слов, содержащихся в строке, и общего количества слов, предполагая, что строка не пуста (есть еще одно слово, которое мы написали, чтобы убедиться в этом). Теперь: если слово, которое мы пишем, должно ожидать в стеке имени файла, буфера, направления двух векторов и количество элементов векторов (равным количеству строк в файле) мы оказываемся с пятью значениями в стеке, что слишком много и привело бы к ситуации в котором большая часть кода будет связана с изменением порядка стека. К Это называется стековый шум . Кроме того, нам придется управлять тремя значения, возвращаемые c-словами. Поэтому предлагается другое решение.

Один из способов упростить задачу — написать
\ d'' id d' d' 0 d \ д'' ид д' д' л \ d'' id d' v \ id д' д''
слово, которое принимает номер столбца в качестве аргумента и преобразует добавьте этот столбец в вектор. Двумя проходами в файл помещаем каждый столбец на своем месте. Другая возможность — переместить файл в матрица. Извлечь векторы-столбцы легко. Более того, мы не ограничено двумя столбцами.

```
: f-a-matriz ( d d’ d’’ --) \ make-array  файл, буфер, массив
  rot dup c-le@
  r/o open-file drop \ d’ d’’ id
  over m-f 0 do \ d’ d’’ id
  rot 2dup \ d’’ id d’ id d’
  dup c-l@ rot \ d’’ id d’ d’ l id
```

```
  read-line \ d'' id d' n flag 0
  drop drop over c-le! \ d'' id d'
  dup c-bordes \ d'' id d' d'
  dup c-#p 0 do
  c-ps \ d'' id d' e
  dup dup c-lp \ d'' id d' e e l
  c-a-entero \ d'' id d' e v
  swap >r >r rot \ id d' d''
  r> over ! cell+ \ id d' d''+c
  -rot r> \ d'' id d' e
loop
 drop -rot
loop close-file 2drop ;
```

6. Слово, которое мы напишем, ожидает в стеке имя файла,
\ д' д'' ид
\ д'' ид д' ид д' \ d'' id d' d' l id \ d'' id d' nf

буфер, в который будут помещены строки, и подстрока, которая будет фильтроваться. Отфильтрованные строки просто распечатываются. Результат может перенаправить в другой файл.

```
: f-grep ( d d' d'' --)
  cr
  rot dup c-le@ r/o
  open-file drop \ d' d'' id
  begin
      rot 2dup \ d'' id d' id d'
      dup c-l@ rot \ d'' id d' d' l id
      read-line drop \ d'' id d' n f
  while
      over c-le! \ d'' id d'
      rot 2dup c-subc \ id d' d'' f
  if
      over c-type cr \ id d' d''
  then
  rot
  repeat drop
  swap close-file 2drop ;
```

7. Теперь у нас есть слово, которое читает столбцы файла.
целые числа и передает их в массив. Но это упражнение другое. потому что не указано, что столбцы должны содержать числа. Наша реализация предполагает, что все строки будут содержать одинаковое количество столбцов. Он ожидает в стеке номера столбцов, буфер, из которого будут считываться строки, и имя файла.

Идея состоит в том, что если a — число в первом столбце, a b — число во втором, прочитайте каждую строку в буфере, перейдите к слову a, распечатайте его и затем перейдите к столбцу, чтобы распечатать его. Для этого у нас есть к-пс

```
: f-list2col ( a b d' d --)
  cr
  dup c-le@ r/o
  open-file drop \ a b d' id
  begin
      2dup over c-l@ swap \ a b d' id d' l id
      read-line drop \ a b d' id n f
  while
      rot 2dup c-le! nip \ a b id d'
      swap >r c-bordes \ a b d'
      rot 2dup \ b d' a d' a
  0 do c-ps loop \ b d' a d''
      dup c-lp
      type 6 espacios \ b d' a
      -rot swap 2dup \ a d' b d' b
```

```
  0 do c-ps loop \ a d' b d''
     dup c-lp type cr \ a d' b
     swap r> \ a b d' id
  repeat drop close-file
 3drop ;
```

Нам бы хотелось, чтобы столбцы были выровнены. Для этого мы могли бы использовать словарь для строк, который у нас уже есть, или просто
напечатайте столько пробелов после каждого слова, сколько необходимо, чтобы заполнить указанную ширину. Напишем еще одну версию этого второго решения:

```
: f-list2col-alineadas ( a b d' d --)
  cr
  dup c-le@ r/o
  open-file drop \ a b d' id
  begin
     2dup over c-l@ swap \ a b d' id d' l id
     read-line drop \ a b d' id n f
  while
     rot 2dup c-le! nip \ a b id d'
     swap >r c-bordes \ a b d'
     rot 2dup \ b d' a d' a
     0 do c-ps loop \ b d' a d''
     dup c-lp tuck
     type 16 swap - spaces \ b d' a
     -rot swap 2dup \ a d' b d' b
     0 do c-ps loop \ a d' b d''
     dup c-lp type cr \ a d' b
     swap r> \ a b d' id
  repeat drop close-file
  3drop ;
```

Как видно, изменения минимальны.

8. Мы можем использовать процедуру из предыдущего упражнения, чтобы взять текстовый файл, содержащий столбцы, и напечатать их выровненными. Проблема в том, что либо все столбцы будут напечатаны с одинаковой шириной, либо нам придется указывать ширину в коде слова, которое мы пишем.
Ничто из этого не практично, поэтому мы хотим выполнить предыдущий проход к файлу и вычислить максимальную ширину каждого столбца, сохранив данные в векторе, который мы будем использовать позже для печати всех выровненных столбцов и каждого с соответствующая ширина.
Слово ожидает в стеке имя файла, буфер и вектор.
В этой реализации подсчитывается количество слов в каждой строке, поэтому не обязательно, чтобы во всех строках было одинаковое количество слов, хотя необходимо, чтобы вектор имел размерность, большую или равную наибольшему количеству слов. который можно найти в строке. Сначала вектор заполняет с нулями.

```
: f-anchura-columnas ( d d' d'' N --)
  2dup 0v!! drop \ d d' d''
  rot dup c-le@ r/o \
  open-file drop \ d' d'' id ----------------+
  begin \ |
     rot 2dup \ d'' id d' id d' |
     dup c-l@ rot \ d'' id d' d' l id |
     read-line drop \ d'' id d' n f |
  while \ |
     over c-le! \ d'' id d' |
     rot 2dup swap \ id d' d'' d'' d' |
     c-bordes dup c-#p \ id d' d'' d'' d' n |
     0 -rot \ id d' d'' d'' 0 d' n |
  0 do \ id d' d'' d'' 0 d' -----+ |
     c-ps dup c-lp \ id d' d'' d'' 0 d' l | |
     swap >r >r \ | |
```

```
      2dup cells + \ id d' d'' d'' 0 d''+c | |
      dup @ r> max \ id d' d'' d'' 0 d''+c v | |
      swap ! 1+ r> \ id d' d'' d'' 0 d' -----+ |
  loop \ |
  3drop rot \ d' d'' id ----------------+
repeat 2drop close-file drop ;
```

В комментариях к коду мы соединили строки, содержащие состояние стека в начале и конце двух циклов (тот, который зацикливается строки файла и тот, который проходит через слова внутри каждой строки) чтобы было ясно, что состояние стека в начале цикла (так должно быть всегда!) одинаково для каждой итерации. Еще одно наблюдение актуально то, что в стеке можно найти до семи элементов, это слишком много. Теперь главное не это число, а количество элементов, которые обрабатываются в данный момент времени, что гораздо меньше.

9. Теперь мы можем распечатать файл с правильно выровненными столбцами. На первом проходе мы заполняем вектор 4 значение каждого элемента, чтобы оставить разделение между столбцами и столбец и распечатайте, следуя процедуре, которую мы использовали в f-list2col-выровнено

```
: f-listcol ( d d' d'' N --)
  cr 2over 2over
  f-anchura-columnas drop \ d d' d'' 2
  rot dup c-le@ r/o \ d' d'' d l r/o 3
  open-file drop rot \ d'' id d' -------------------------+ 4
  begin \ | 5
      2dup dup c-l@ rot \ d'' id d' d' l id | 6
      read-line drop \ d'' id d' n f | 7
  while \ | 8
      over c-le! \ d'' id d' | 9
      rot swap \ id d'' d' | 10
      2dup \ id d'' d' d'' d' | 11
      c-bordes dup c-#p \ id d'' d' d'' d' n | 12
  0 do \ id d'' d' d'' d' --------------+ | 13
      c-ps dup c-lp \ id d'' d' d'' d' l | | 14
      3dup \ id d'' d' d'' d' l d'' d' l | | 15
      tuck type \ d'' l | | 16
      swap @ 4 + swap - \ | | 17
      spaces drop \ id d'' d' d'' d' | | 18
      swap 1 cells + \ | | 19
      swap \ id d'' d' d'' d' --------------+ | 20
  loop cr \ | 21
  2drop rot swap \ d'' id d'-------------------------+ 22
  repeat \ 23
  2drop close-file 2drop ; \ 24
```

Опять же, мы наблюдаем, что в куча. Однако вы никогда не оперируете более чем тремя. Во внешнем цикле Идентификатор файла и адреса файлов должны быть сохранены. буфер и вектор. Во внутреннем цикле необходимо работать с буфером и вектор, но они также должны сохраняться, увеличиваясь между итерация и следующая. Следовательно, 2dup строки 11 и 3dup строки 15. Мы соединили строки 4-22 и 13-20, чтобы показать, что состояние стека одинаково на каждой итерации. Однако это длинное слово, которое можно было бы учесть лучше. Например, написав независимое слово для печати строки, которая воспринималась как аргументы — направление строки и вектора. мы бы устранили таким образом, внутренний цикл.

2.9. Клавиатура и экран

1.
```
: at-fc swap at-xy ;
: t-imprc ( ca f co --)
  at-fc emit ;
```

2.
```
: t-imprc* ( ca f co n --)
```

```
  -rot at-fc 0 do
  dup emit
loop drop ;
```

3.

```
: t-ascii ( --)
  cr 6 spaces
  32 128 32 do
  dup 4 .r space
  dup emit
  dup 10 mod 0= if cr then
  1+
loop cr ;
```

4.

```
: t-horizontal ( f c ancho --)
  >r 2dup at-fc [char] + emit r>
  swap 1+ swap
  2 - 0 do
      2dup at-fc [char] - emit
      1+
  loop at-fc [char] + emit ;

: t-vertical ( f c alto --)
  0 do
      2dup at-fc [char] | emit
      swap 1+ swap
  loop 2drop ;

: t-cuadrado ( f c ancho alto --)
  >r 3dup t-horizontal
  3dup rot r> dup >r + 1- -rot t-horizontal
  r> swap >r 3dup rot 1+ -rot 2 - t-vertical
  swap r> + 1- swap rot 1+ -rot 2 - t-vertical ;
```

5. Как только прямоугольник будет нарисован, символ «*» будет перемещаться между строками f+1 и f +height-2 и столбцами c+ 1 и c+width-2. Мы поместим звездочку в рамку, верхний левый угол которой имеет координаты i (0, 0) и ширину и высоту 20 символов. Таким образом, звездочка будет перемещаться между строками 1 и 18 включительно и столбцами 1 и 18 включительно.

```
: *on ( f c --) at-fc [char] * emit ;
: *off ( f c --) at-fc 32 emit ;
: t-prision ( --)
  page
  0 0 20 20 t-cuadrado
  10 10 2dup *on
  begin
      key
      dup 113 <> \ 'q'
  while
      dup 97 = if \ 'a' izquierda
       drop dup 1 > if
            2dup *off 1- 2dup *on
       then
      else
  dup 115 = if \ 's' derecha
   drop dup 18 < if
     2dup *off 1+ 2dup *on
   then
  else
   dup 119 = if \ 'w' arriba
   drop over 1 > if
    2dup *off swap 1- swap 2dup *on
   then
```

```
  else
    dup 122 = if \ 'z' abajo
    drop over 18 < if
    2dup *off swap 1+ swap 2dup *on
  then
 else
drop
then then then then
repeat 2drop page ;
```

6. И терминал ANSI, и терминал Linux распознают серию escape-последовательностей , которые представляют собой не что иное, как строки символов, первой из которых является символ ESC (27). Некоторые из этих последовательностей позволяют изменять атрибутытекста, представленного на экране. Формат этих строк — ESC[xm, где x — символ: 0 для отключить все атрибуты; 1 для выделенного текста; 4 для текста подчеркнутый; 5 для мерцания и 7 для обратного видео. Это тривиально написать слова, чтобы активировать некоторые из них и деактивировать их:

```
: ESC 27 emit ;
: t-desactiva ( --) ESC s" [0m" type ;
: t-resaltado ( --) ESC s" [1m" type ;
: t-subrayado ( --) ESC s" [4m" type ;
: t-parpadeo ( --) ESC s" [5m" type ;
: t-inverso ( --) ESC s" [7m" type ;
```

7. Сохраним последовательность символов в строке. мыначнем заполнив вектор алфавитной последовательностью, а затем мы i создадим несколько случайных обменов мнениями. Как только это будет сделано, мы ' нарисуем доску и каждуюбукву в своем квадрате, и мы' войдем в цикл, который управлять движениями.

```
: t-juego-horizontal
  s" +---+---+---+---+" type ;
: t-juego-vertical
  s" |   |   |   |   |" type ;
: t-juego-tablero
  page 0 0 at-fc
  4 0 do
      t-juego-horizontal cr
      t-juego-vertical cr
  loop
  t-juego-horizontal cr
  s" W" type cr
  s" A S Ctrl-A" type cr
  s" Z" type ;

: t-juego-casillas ( d --) \ direccion de la cadena
  0 swap \ i d
  16 0 do
      over 4 /mod swap \ i d f c
      4 * 2 + swap \ paso a pantalla
      2 * 1 + swap \ i d f c
      at-fc 2dup + c@ emit \ i d
      swap 1+ swap \ i+1 d
  loop 2drop ;

: rnd0-15 ( a --b b) \ en el rango [0,15]
  899 * 32768 mod 16 mod dup ;
: t-juego-mezcla ( d s --) \ cadena y semilla
  10 0 do
      rnd0-15 rnd0-15 \ d a b b
      >r rot tuck \ a d b d
      dup >r
      + -rot + exch1
      r> r> \ d b
  loop 2drop ;
```

```
: t-juego-inicial ( d --n) \ исходное положение отверстия
  0 \ d 0
  begin
      2dup + c@ [char] . <> \ d 0 f
  while
      1+
  repeat nip ;

: t-juego-izquierda ( d n -- d n-1)
  dup 4 mod 0> if
   2dup 2dup \ d n d n d n
   1- + -rot + exch1 \ d n
   1- \ d n-1
  then ;

: t-juego-derecha ( d n -- d n+1)
  dup 4 mod 3 < if
      2dup 2dup
      1+ + -rot + exch1
      1+
  then ;

: t-juego-arriba ( d n -- d n’)
  dup 4 / 0> if
      2dup 2dup
      4 - + -rot + exch1
      4 -
  then ;

: t-juego-abajo ( d n --d n’)
  dup 4 / 3 < if
      2dup 2dup
      4 + + -rot + exch1
      4 +
  then ;

: t-juego-bucle ( d n --) \ распродажа с Ctrl-A
  begin
      key dup 1 <> \ d n k f
  while \ d n k
      dup 97 = if \ d n k
        drop \ d n
        t-juego-derecha \ d n+1
      else
       dup 115 = if
            drop
            t-juego-izquierda
        else
       dup 119 = if
            drop
            t-juego-abajo
       else
      dup 122 = if
            drop
            t-juego-arriba
       else
      drop
then then then then
over t-juego-casillas
repeat 3drop ;

: t-juego ( s -- ) \
  page t-juego-tablero
  s" diklbhjagmcenof." \ s d n
```

```
  drop tuck
  swap t-juego-mezcla \ d
  dup t-juego-casillas \ d
  dup t-juego-inicial \ d n
  t-juego-bucle
  page ;
```

8. Будем опираться на битовые комбинации биоса, где они закодированы каждая фигурка каждого персонажа. Мы записали небольшую программу на C под DOS, которая обращается к таблице ПЗУ, где расположенывти шаблоны, и сохранили их в файле, который мы назвали Patterns.dat.

Этот файл содержит в общей сложности 4096 строк, которые представляют собой 256 блоков по 16 строк.

Каждая строка содержит число, поэтому символ, заданный 16 байтами, можно рассматривать как массив из 16 строк и 8 столбцов. Биты, установленные в 1, обозначают освещенные пиксели.

Чтобы напечатать макроверсию символа, мы сначала загружаем Patterns.dat в вектор (слово fa-vector у нас уже есть).
Таким образом, номер символа ASCII n кодируется, начиная с позиции 16n.

```
: t-impr1 ( n --) \ вывести байт
  7 0 do
     dup 2 mod
     swap 2 /
  loop
  8 0 do
     1 = if
           [char] # emit
     else
           32 emit
     then
loop cr ;

: t-impr16 ( d --) \ учитывая адрес
  \ печатает содержимое и
  \ следующее 15 байтов
  16 0 do
     dup @ t-impr1
     1 cells +
  loop drop ;

: t-cargar-patrones ( d d' d'' N --)
  f-a-vector ;

: t-grande ( c d'' --)
  swap 16 * cells +
  t-impr16 ;
```

9. Слово, которое мы i собираемся записать, должно ожидать в стеке: вектор с битовыми комбинациями символов и адресом строки, которая собирается печатать. Для простотыстрока будет отображаться в первом терминальнуюстроку и представление каждого символа из строка 3 и столбец 0.

```
: t-deletrea0 ( c d'' --)
  3 0 at-fc t-grande ;

: t-deletrea ( d d'' N --) \ удаление страницы
  page drop
  over c-type 1 0 *on \ d d''
  over c@ over \ d d'' c d''
  t-deletrea0 \ d d''
  swap 0 \ d'' d contador
```

```
  begin
      key dup 1 <> \ d'' d contador k f
  while \ d'' d contador k
      dup 97 = if \ d'' ожидаем строку и шаблоны
       drop \ d'' d contador
       dup 0 > if
            dup 1 swap *off
            1- dup 1 swap *on
            2dup + c@ \ d'' счетчик c
            >r rot dup r> swap \ d счетчик d'' c d''
            t-deletrea0 -rot \ d'' счетчик
       then
      else
            dup 115 = if \ d'' счетчик k
                 drop \ d'' d contador
                 over c-le@ \ d'' счетчик l
                 over > if \ d'' счетчик
                      dup 1 swap *off
                      1+ dup 1 swap *on \ d'' счетчик
                      2dup + c@ \ d'' счетчик c
                      >r rot dup r> swap \ счетчик d'' c d''
                      t-deletrea0 -rot
                 then
            else
      then then
  repeat 2drop 2drop ;
```

## 2.10. Структуры данных

### 2.10.1. Стеки и очереди

1. Гри создании в первой ячейке указывается индекс первой свободной позиции, а во второй - максимальное значение индекса. Таким образом, можно узнать, разряжена или заряжена батарея. Во время выполнения сохраняется адрес первого элемента стека.

```
: pila create 0 , 3 - dup , cells allot
  does> 2 cells + ;
```

2. Также необходимо будет написать слова, обозначающие, пуста ли стопка или полна. Что касается слов, которые складываются и разбираются, то в результате необходимо будет оставить флаг, указывающий, прошла операция успешно или нет. Наконец, p-top возвращает адрес следующей свободной ячейки, p-i+ увеличивает индекс, а pilo уменьшает.
В целях отладки удобно написать еое одно слово, которое печатает элементы i стека, и другое, которое очищает стек.

```
: p-vacia? ( d --f)
  2 cells - @ 0= ;
: p-llena? ( d --f)
  dup 2 cells - @
  swap 1 cells - @ > ;
: p-# ( d --n) \ количество сложенных элементов
  2 cells - @ ;
: p-tope ( d --d')
  dup 2 cells - @ cells + ;
: p-i+ ( d --)
  2 cells - dup @ 1+ swap ! ;
: p-i- ( d --)
  2 cells - dup @ 1- swap ! ;
: >p ( n d --) \ небезопасно
  tuck p-tope ! p-i+ ;
: p> ( d -- n) \ небезопасно
  dup p-tope 1 cells - @
  swap p-i- ;
```

```
: p. ( d --)
  dup p-vacia? if drop exit then
  cr dup 2 cells - @
  0 do
     @+c . cr
  loop ;
: p-0 ( d --)
  2 cells - 0 swap ! ;
```

3.

```
: p-drop ( d --)
  p> drop ;
: p-dup ( d --)
  dup p-tope 1 cells - @
  swap >p ;
: p-swap ( d --)
  dup dup p> swap p>
  >r over >p r> swap >p ;
```

4. Мы реализуем очередь FIFO, используя 13 слов. Грежде всего, создать очередь:

```
: fifo create dup , 0 , 0 , 0 , 4 - allot
  does> 4 cells + ;
```

Гри создании первые четыре ячейки зарезервированы: одна для обиее количество элементов, другой для текущего количества элементов, другой для указатель чтения и четвертый для указателя записи.

5. Двенадцать слов для управления очередью:

fifo-L? возвращает указатель чтения;
fifo-E? вернуть указатель записи;
fifo-H? возвраиает максимальное количество элементов, которые может
 хранить очередь.
fifo-#? возвращет количество элементов, фактически содержащихся в
 очереди;
fifo-L выполняет чтение;
fifo-E выполняет запись;
fifo-#+ увеличивает количество элементов;
fifo-#- декрементирует количество элементов;
fifo-L+ увеличивает указатель чтения; ;
fifo-E+ увеличивает указатель записи.

Операции чтения В стеке оставляют: флаг, установленный в ноль, если очередь пуста;
считанное значение и установите флаг в -1 в противном случае.
Операции записи потребляют ваше аргумент в любом случае и оставьте флаг 0 или -1 в зависимости от операции не удалось или удалось.

Нам будут полезнылеие два дополнительных слова. скажем, полна ли очередь или пуста, и треть для целей отладка, печатающая склеенные элементы.

```
: fifo-L? ( d -- n)
  2 cells - @ ;
: fifo-E? ( d --n)
  1 cells - @ ;
: fifo-N? ( d --N)
  4 cells - @ 4 - ;
: fifo-#? ( d --n)
  3 cells - @ ;
: fifo-L+ ( d --)
  2 cells - dup dup @ 1+ swap
  fifo-N? mod swap ! ;
: fifo-E+ ( d --)
  1 cells - dup dup @ 1+ swap
```

```
  fifo-N? mod swap ! ;
: fifo-#+ ( d --)
  3 cells - dup @ 1+ swap ! ;
: fifo-#- ( d --)
  3 cells - dup @ 1- swap ! ;
: fifo-llena? ( d -- f)
  dup fifo-N? swap fifo-#? = ;
: fifo-vacia? ( d -- f)
  fifo-#? 0= ;
: fifo-L ( d --n)
  dup fifo-vacia? if
       drop 0
  else
   dup dup fifo-L? cells +
   @ swap -1 swap
   dup fifo-L+ fifo-#-
  then ;
: fifo-E ( n d --)
  dup fifo-llena? if
    2drop 0
  else
    dup dup fifo-E? cells +
    rot swap !
    dup fifo-E+ fifo-#+ -1
  then ;
: fifo. ( d --)
  cr dup dup fifo-L? swap fifo-E?
  begin
     2dup <
  while
     -rot 2dup cells + @ . cr
     1+ rot
  repeat drop ;
```

### 2.10.2. Дружественная динамическая память

Цель этого раздела — написать менеджер динамической памяти, который будет использоваться позже при работе с другими структурами данных, такими как списки и деревья. Фрагмент памяти, которым нужно управлять, берется из самого словаря. Что касается самого управления, то оно будет осуществляться через пару связанных списков: один для свободных фрагментов, другой для занятых.

При запросе памяти список свободных фрагментов просматривается до тех пор, пока не будет найден фрагмент, превышающий запрошенное количество. Затем этот фрагмент делится на два: одна часть резервируется и добавляется в список занятых, а другая остается в списке свободных, обновляя его размер. Освободить память просто: фрагмент перемещается из одного списка в другой.
Каждый фрагмент содержит не только объем свободной или занятой памяти, но и указатель на следующий фрагмент того же класса и размера.
Наконец, из всего количества, подлежащего управлению, необходимо будет зарезервировать четыре ячейки:
два указателя на начало и конец списка свободных мест и еще два — на список занятых фрагментов.
Подготовка места в словаре производится с помощью следующих слов:

```
: almacen create
  here 4 cells + dup , ,
  0 , 0 ,
  0 , 6 cells - ,
  allot
  does> ;
```

Как видите, адреса, сохраненные в заголовках, являются абсолютными адресами.
Создание хранилища начинается с создания словарной записи и компиляции указателей, где 0 равно нулю.

Затем пространство делится на первую часть с шестью ячейками и вторую часть с остальными до тех пор, пока не будет достигнут указанный размер. В первой части четыре ячейки обозначают первый свободный блок, последний свободный блок, первый занятый блок и последний занятый блок. Еще двое — это заголовок первого свободного блока, который при запуске хранилища является единственным существующим и имеет тот размер, который был указан за вычетом шести именованных ячеек. Чтобы избежать конфликтов имен, мы будем использовать префикс md- во всех словах. Адрес магазина обозначается в комментариях стека буквой «a».

```
: md-PL ( a --d)
  @ ;
: md-UL ( a --d)
  1 cells + @ ;
: md-PO ( a --d)
  2 cells + @ ;
: md-UO ( a --d)
  3 cells + @ ;
```

PL указывает адрес первого свободного блока. UL адрес последнего свободного блока. PO — адрес первого занятого блока, UO — адрес последнего занятого блока. +

```
+-----------------------------------------
|16 |16 | 0 | 0 | 0 | t | | ...
+-----------------------------------------
\_____________/ \_____/
| | заголовок
| +--> первый блок
|
+--> заголовок склада:
4 celdas = 16 bytes
```

Что касается данного блока, интерес представляет адрес первого используемого байта. Учитывая этот адрес, следующие два слова получают размер блока и смещение следующего блока в его списке:

```
: md-S ( d -- d’)
  2 cells - @ ; \ следующий

: md-T ( d --t)
  1 cells - @ ; \ taman~o
```

Следующее, что нам нужно попробовать, это по запросу пройтись по списку свободных блоков, пока не найдем один, достаточно большой, чтобы его можно было разделить на две части: часть, которая будет добавлена в список занятых, и другая часть, которая останется в свободном. , но с измененным размером. Как часть, которая занимается
Он также должен содержать в заголовке, который представляет собой две ячейки, свободный размер должен превышать запрошенный размер на две ячейки.
Итак, напишем два слова: одно, которое ищет достаточно большое свободное место, и другое, которое выполняет деление и обновляет списки. md-hole ожидает в стеке размера и адреса хранилища и ищет блок из этого хранилища для его размещения. Если найден, возвращает адрес блока. Если нет, возвращает 0. md-fit? ожидает размер и адрес блока и указывает, может ли этот размер разместиться в блоке.

```
: md-cabe? ( t d --f)
  md-T 2 cells - < ;

: md-hueco ( t d -- d’ )
  2dup md-cabe? if
  nip
  else
   dup 0= if
   nip
  else
```

```
   md-S recurse
  then
 then ;
```

После того, как найден блок, способный вместить запрошенный размер, необходимо:

Разделите его на две части, создав заголовок нового занятого блока.
Добавьте этот новый занятый блок в список занятых блоков.
Обновите указатель на 'последний занятой блок .
Обновите размер исходного свободного блока оставшимся значением.
после вычитания запрошенного размера
Слово md-reserve ожидает в стеке адрес склада, адрес свободного блока, который может вместить запрошенное резервирование, и размер, который требуется зарезервировать.

```
: md-reserva ( a d r --) \ 1
  dup >r \ para despues 2
  2dup over md-T swap - + \ a d r d' 3
  tuck 1 cells + ! \ a d d' 4
  0 over ! \ a d d' 5
  rot dup md-UO >r \ 6
  -rot dup r> ! \ a d d' 7
  rot 3 cells + over swap ! \ d d' 8
  swap r> \ d' d r 9
  over md-T swap - 2 cells - \ 10
  swap 1 cells + ! ; \ d' 11
```

В строке 3 вычисляется адрес нового занятого блока. В 4-м заполняет вторую ячейку заголовка размером. В строке 5 заполняется первая ячейка заголовка, указывающая на следующий занятый блок. Поскольку создаваемый блок является последним, указатель устанавливается в 0. В строки 6-7
получаются из заголовка магазина по адресу последнего занятый блок и указывает на вновь созданный блок, который становится последним. В строке 8 обновляется указатель на последний занятый блок в заголовке. со склада. Наконец, свободный размер обновляется в блоке, из которого он взят. принял бронь.
Следующая базовая операция состоит в освобождении блока. Для Этого достаточно, чтобы перевести его из списка занятых в список свободных, добавив в список конец этому. В свою очередь, этот перенос состоит из нескольких элементарных операций:

Найдите блок перед тем, который нужно освободить.
Свяжите это с блоком, следующим за тем, который мы хотим выпустить.
Перенести занятый блок в список свободных блоков.
мд-первый-занят? проверяет, является ли блок, который вы хотите освободить первый в списке, возвращающий флаг. мд-предыдущий-занят? ждать два адреса: один из списка, другой из блока, который вы хотите освободить, и возвращает адрес блока, предшествующего тому, который вы хотите освободить. Работает рекурсивно. md-search ожидает адрес склада и адрес блок, который вы хотите освободить. Если этот блок первый, то адрес склада. Если адрес г. первый занятый блок и адрес того, который вы хотите занять и найти отсюда и предыдущий блок с использованием предыдущей рекурсивной функции.

```
: md-primero-ocupado? ( a d --f)
     swap md-PO = ;

: md-anterior-ocupado? ( d d' -- d'')
  2dup swap md-S =
  if
     drop
  else
  swap md-S
  swap recurse
  then ;

: md-busca ( a d -- d')
  2dup md-primero-ocupado?
```

```
  if
      drop
  else
      swap md-PO
      swap md-anterior-ocupado?
  then ;
```

Связать предыдущий блок с тем, который вы хотите выпустить, со следующим легко. md-enlaza ждет адреса предыдущего блока и адреса того, который вы хотите освободить.

```
: md-enlaza ( d d' --)
  md-S swap ! ;
```

Наконец, освобожденный блок должен быть интегрирован в список свободных. md-transfer берет адрес занятого блока и связывает его с последним в свободном списке. Также дождитесь адреса склада. Необходимо: установить ссылку блока, который передается в 0, связать последний в свободном списке с тем, который передается и изменить в заголовке склада в поле, соответствующем последнему элементу в бесплатный список.

```
: md-traspasa ( a d --)
  dup 0 swap !
  over md-UL over
  swap ! swap ! ;
```

Не хватает двух вещей: создать с помощью этих слов интерфейс malloc и free , который сводится к небольшому количеству c'odigo Mortor , и написать дефрагментатор, поскольку в случае сбоя malloc он может дефрагментировать память, и запрос может быть удовлетворен. . Фактически, если хранилище будет исчерпано серией запросов небольшого размера, а затем будет освобожден каждый фрагмент, мы окажемся в точке, где хранилище полностью свободно, но нет блока, который можно было бы использовать. Именно поэтому необходимо написать дефрагментатор, объединяющий свободные смежные блоки в один.
Дальше продолжать не будем по нескольким причинам: 1) чтобы не усложнять простое упражнение; 2) потому что в системах уже есть менеджер динамической памяти; 3) потому что это увело бы нас дальше, чем мы хотим определить переменные типа указателя и определить операции с указателями, и 4) потому что мы только что увидели, как управлять списком.
Фактически: можно использовать указатели и универсальный менеджер памяти для создания списков и управления ими, или можно писать непосредственно в пространстве, заданном менеджером списков. Это то, что мы сделали для создания менеджера памяти. И очевидно, что то, что мы сделали для одной цели, можно сделать и для другой.